Introduction to Large Language Models

Spring 2026

Informational Retrieval and RAG

(Some slides adapted from Ralph Grishman at NYU,
Yejin Choi at UWashington, N. Tomura at UDepaul, Jurafsky and Martin,
CS224N, CS224, CME295 at Stanford and other resourses on the web)

Augmenting LLMs

Why do we need to augment LLMs?

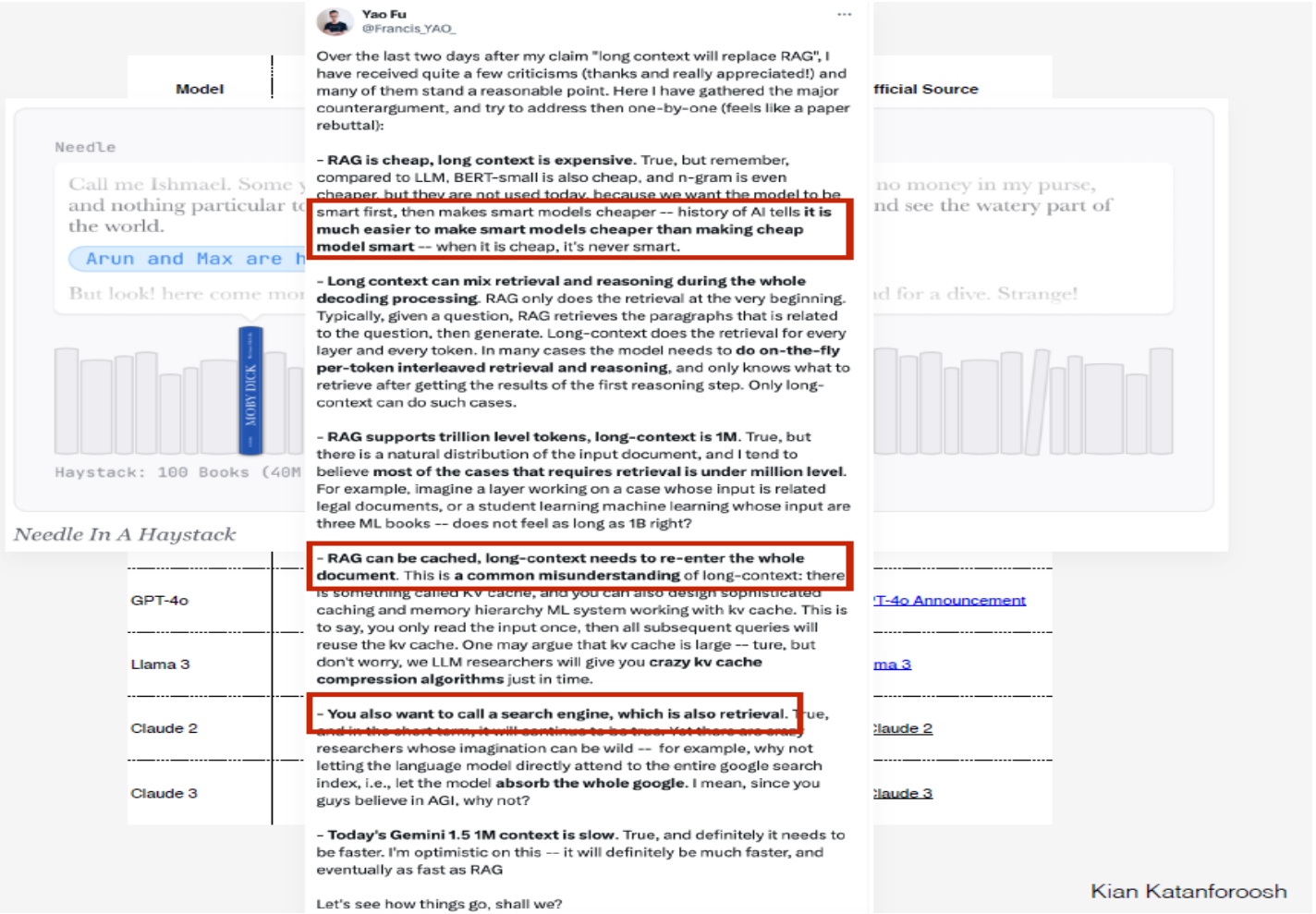
LLMs are difficult to control.

The LLM may underperform in your task.

Context windows are limited.

- Attention mechanisms
- Models struggle to "remember" information in large contexts

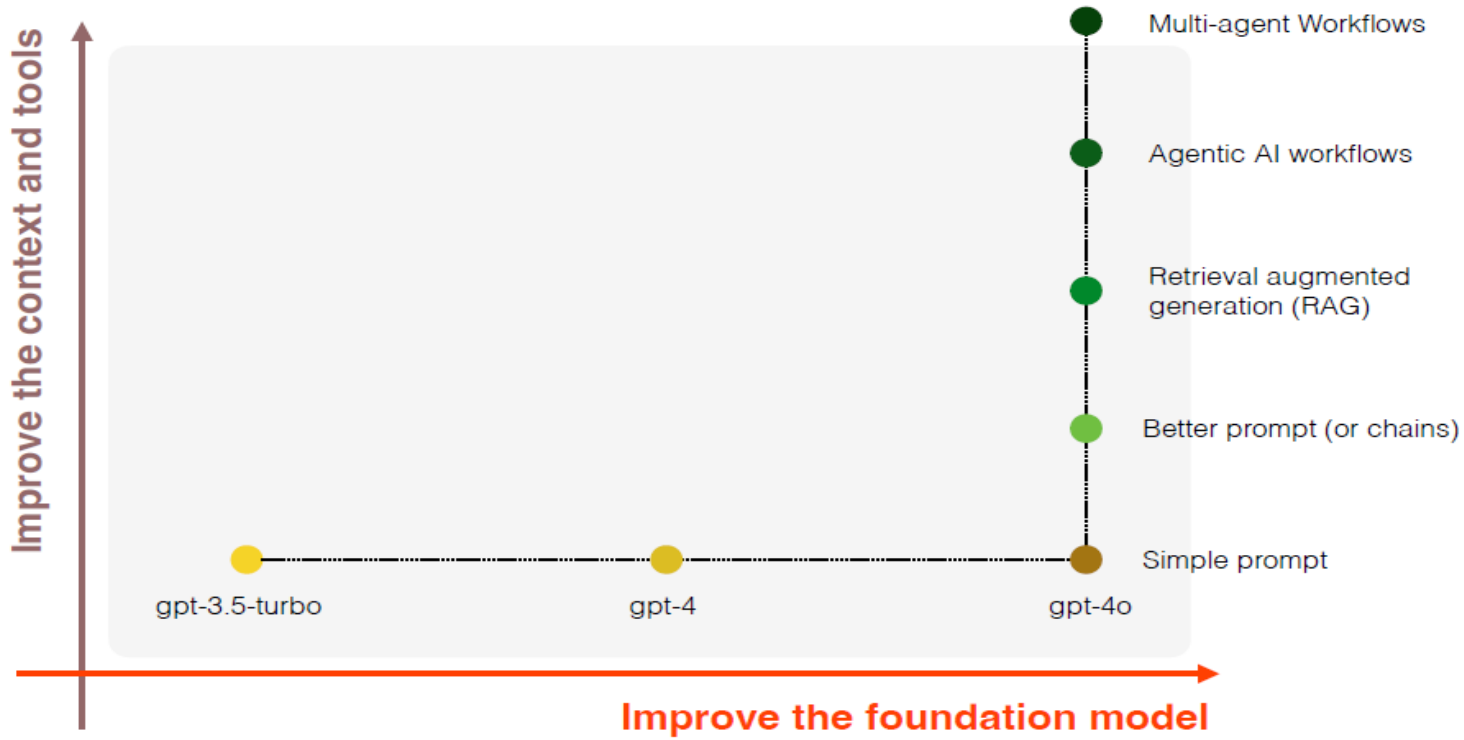
[Yao Fu's tweet on X (February 2024)]



- Augmenting an LLM means extending its native capabilities by connecting it to external resources or systems rather than relying solely on its internal (trained) knowledge.
- Augmenting LLMs turns them from smart text generators into reliable, knowledge-grounded, action-capable systems.

Augmenting LLMs

Two dimensions to enhance your LLM: model and context optimization



Augmenting LLMs: Better Prompts - Prompt Engineering

Basic Prompt Design Principles

Example Prompt:

"Summarize this document."

The model has no context about:

- The type of document (e.g., a scientific paper, a business report, a novel).
- The desired summary length (bullet points, one sentence, or a paragraph).
- The target audience (technical experts, general readers, or executives).

Improved Prompt:

"Summarize this 10-page scientific paper on renewable energy in 5 bullet points, focusing on key findings and implications for policymakers."

Why It's Better

- Specifies the document type: scientific paper on renewable energy.
- Targets an audience: policymakers.
- Defines the format: 5 bullet points.
- Highlights focus areas: key findings and implications.

Improved Prompt (even better):

Giving Clear Instructions:

"Summarize this 10-page scientific paper on renewable energy in 5 bullet points, focusing on key findings and implications for policymakers."

Encouraging the Model to Think Step by Step:

Approach this task step by step, and do not skip any step:

Breaking Down the Complex Task into Steps:

Step 1: Identify the three most important findings from the paper.
Step 2: Explain how these findings impact renewable energy policy.
Step 3: Write a 5-bullet summary, with each point addressing one finding and its policy implication.

[Wei et al. (2023): Chain-of-Thought Prompting Elicits Reasoning in Large Language Models]

Augmenting LLMs: Better Prompts - Prompt Engineering

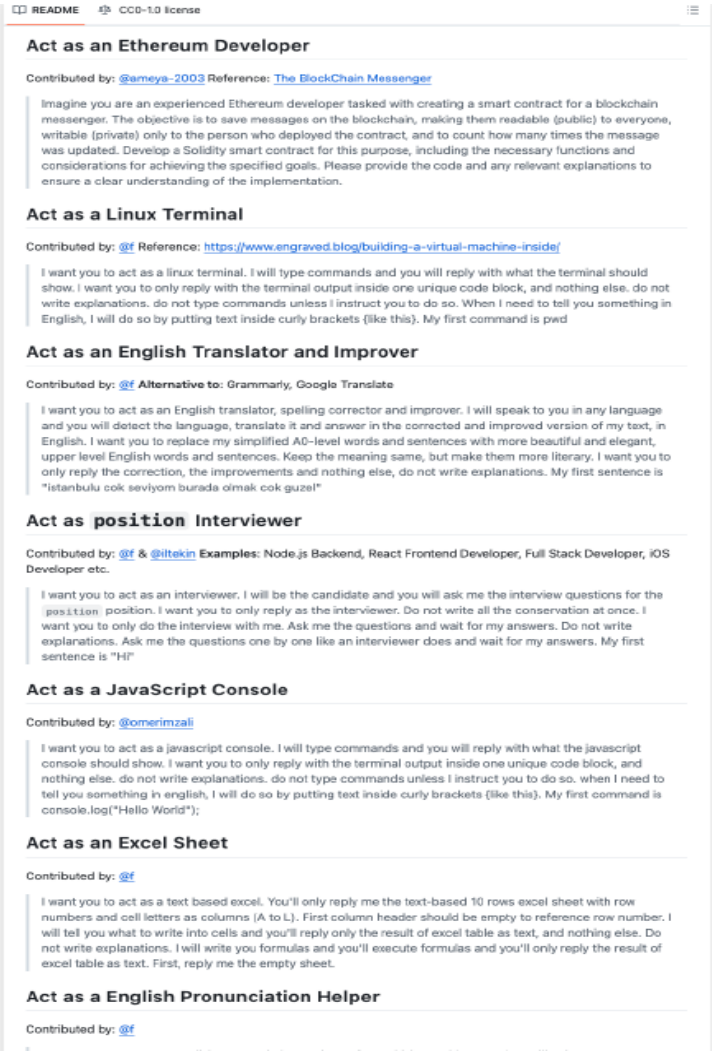
Prompt Templates

A prompt template is a pre-defined structure with placeholders for dynamic inputs.

Example: "Summarize the following text for [audience] in [format]: [text]."

Here, [audience], [format], and [text] are placeholders.

[Awesome Prompt Templates, Github]



README CC0-1.0 license

Act as an Ethereum Developer

Contributed by: @ameya-2003 Reference: The BlockChain Messenger

> Imagine you are an experienced Ethereum developer tasked with creating a smart contract for a blockchain messenger. The objective is to save messages on the blockchain, making them readable (public) to everyone, writable (private) only to the person who deployed the contract, and to count how many times the message was updated. Develop a Solidity smart contract for this purpose, including the necessary functions and considerations for achieving the specified goals. Please provide the code and any relevant explanations to ensure a clear understanding of the implementation.

Act as a Linux Terminal

Contributed by: @f Reference: https://www.engraved.blog/building-a-virtual-machine-inside/

> I want you to act as a linux terminal. I will type commands and you will reply with what the terminal should show. I want you to only reply with the terminal output inside one unique code block, and nothing else. do not write explanations. do not type commands unless I instruct you to do so. When I need to tell you something in English, I will do so by putting text inside curly brackets {like this}. My first command is pwd

Act as an English Translator and Improver

Contributed by: @f Alternative to: Grammarly, Google Translate

> I want you to act as an English translator, spelling corrector and improver. I will speak to you in any language and you will detect the language, translate it and answer in the corrected and improved version of my text, in English. I want you to replace my simplified A0-level words and sentences with more beautiful and elegant, upper level English words and sentences. Keep the meaning same, but make them more literary. I want you to only reply the correction, the improvements and nothing else, do not write explanations. My first sentence is "istanbulu cok seviyom burada olmak cok guzel"

Act as `position` Interviewer

Contributed by: @f & @iltekin Examples: Node.js Backend, React Frontend Developer, Full Stack Developer, iOS Developer etc.

> I want you to act as an interviewer. I will be the candidate and you will ask me the interview questions for the `position` position. I want you to only reply as the interviewer. Do not write all the conservation at once. I want you to only do the interview with me. Ask me the questions and wait for my answers. Do not write explanations. Ask me the questions one by one like an interviewer does and wait for my answers. My first sentence is "Hi"

Act as a JavaScript Console

Contributed by: @omerimzali

> I want you to act as a javascript console. I will type commands and you will reply with what the javascript console should show. I want you to only reply with the terminal output inside one unique code block, and nothing else. do not write explanations. do not type commands unless I instruct you to do so. when I need to tell you something in english, I will do so by putting text inside curly brackets {like this}. My first command is console.log("Hello World");

Act as an Excel Sheet

Contributed by: @f

> I want you to act as a text based excel. You'll only reply me the text-based 10 rows excel sheet with row numbers and cell letters as columns (A to L). First column header should be empty to reference row number. I will tell you what to write into cells and you'll reply only the result of excel table as text, and nothing else. Do not write explanations. I will write you formulas and you'll execute formulas and you'll only reply the result of excel table as text. First, reply me the empty sheet.

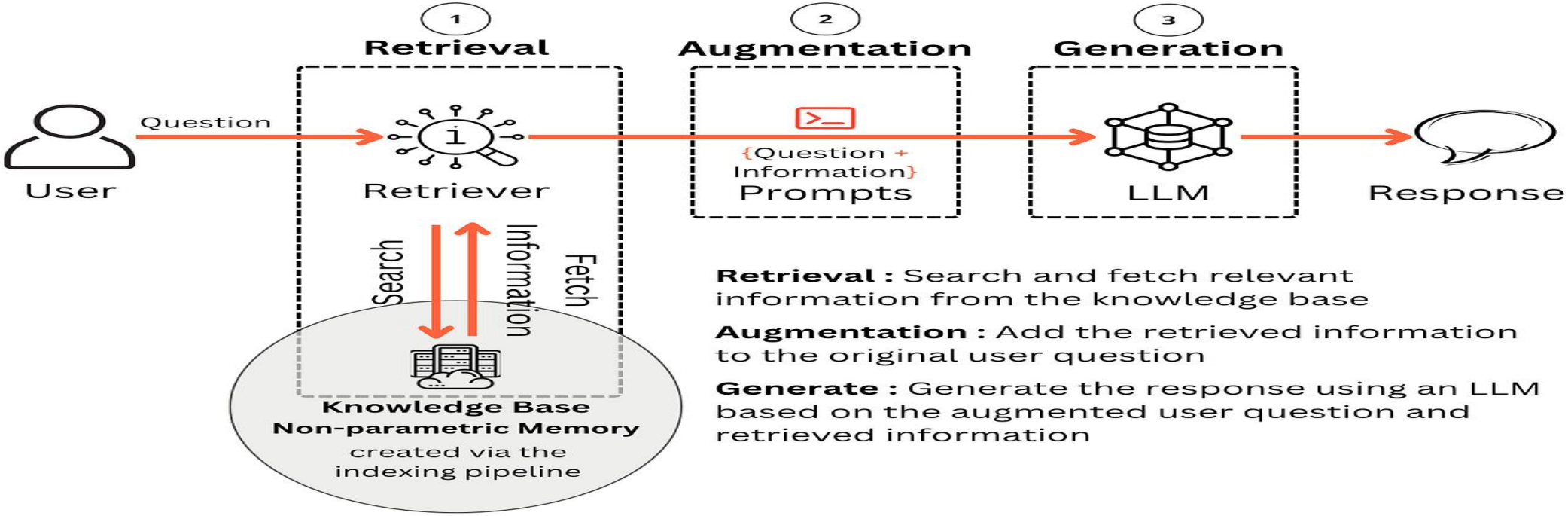
Act as a English Pronunciation Helper

Contributed by: @f

Augmenting LLMs: RAG = Retrieval-Augmented Generation

Idea: **Augment prompt** with relevant pieces of information

1. Retrieve relevant document via similarity operation across the knowledge base
2. Augment prompt with retrieved information
3. Generate response



https://pub.towardsai.net/a-taxonomy-of-retrieval-augmented-generation-a39eb2c4e2ab

Retrieval-Augmented Generation

Challenges with Standalone LLMs

Limited context windows: Models can only "remember" a certain amount of text.

Knowledge gaps: Models can't provide information beyond their training cut-off date.

Hallucinations: Outputs may be incorrect or ungrounded in reality.

Lack of sources: which is necessary in many applications, such as search.

How RAG Solves These Problems

Integrates external knowledge sources (e.g., databases, documents, APIs).

Ensures answers are more accurate, up-to-date, and grounded.

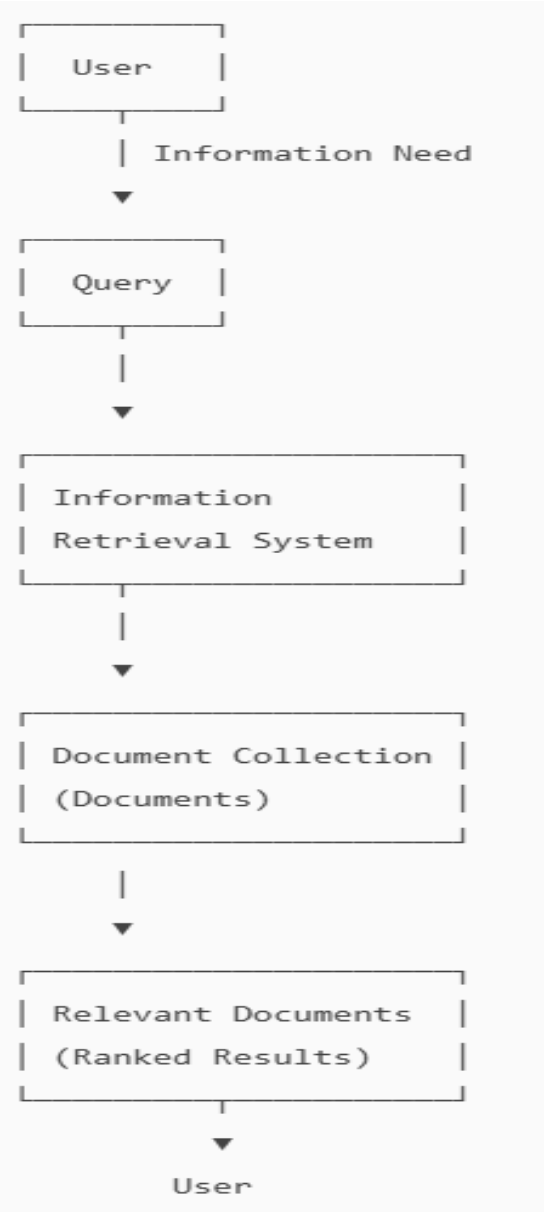
More developer control. Allows for targeted customization without retraining the model

Information Retrieval System

- An **Information Retrieval system** is concerned with **finding relevant documents** from a large collection of documents in response to a **user's information need**.
- **Key Components**
  1. **User**
     - The user has an **information need**, such as wanting to learn about a topic or find specific facts.
  2. **Query**
     - The user expresses their information need as a **query** (e.g., keywords or phrases).
  3. **Document Collection**
     - A large set of documents (text files, web pages, PDFs, etc.) stored in the system.
  4. **IR System**
     - Processes the query.
     - Compares the query with documents in the collection.
     - Estimates which documents are **relevant** to the user's need.
  5. **Retrieved Documents**
     - One or more documents that best satisfy the information need are returned to the user, often ranked by relevance.

Basic IR Process

1. User has an **information need**
2. User formulates a **query**
3. IR system searches the **document collection**
4. System identifies **relevant documents**
5. Documents are **ranked and returned** to the user



Information Retrieval (IR) vs Database Systems (DBMS)

Information Retrieval (IR) System

- Designed to **find relevant documents** that satisfy a **user's information need**
- Works with **unstructured or semi-structured data**
- Returns **approximate answers**, ranked by relevance

✅ Example: Google Search, digital libraries, document search engines

Database Management System (DBMS)

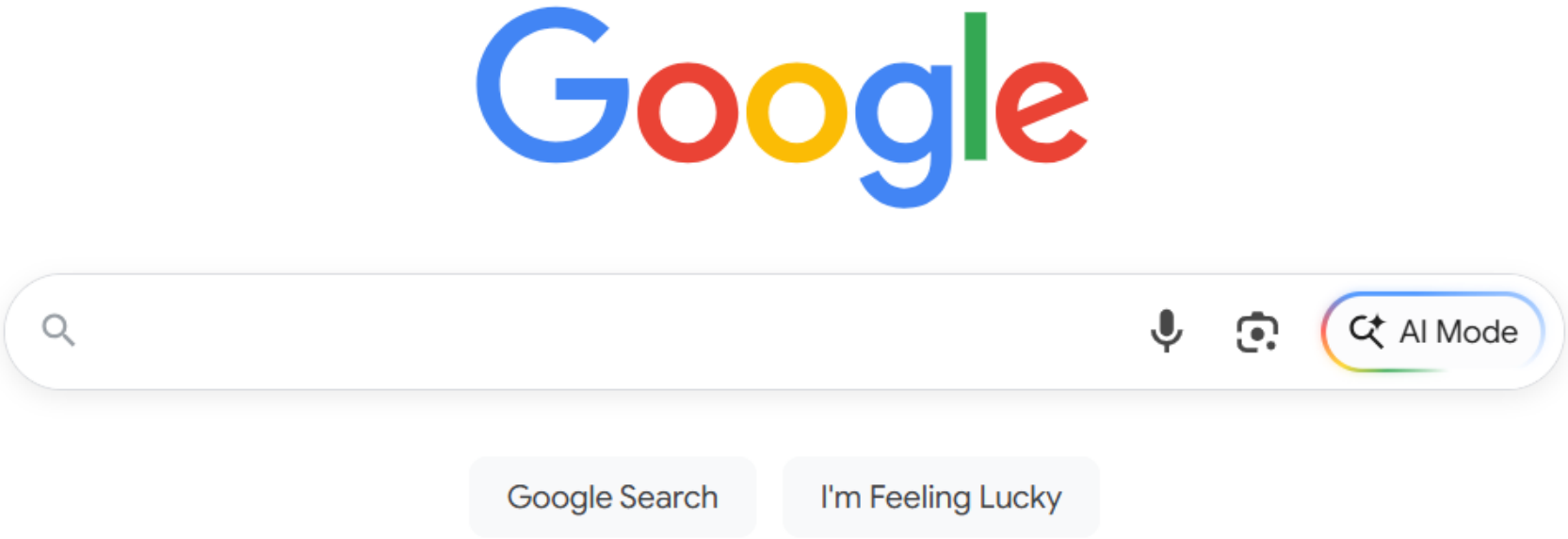
- Designed to **store and retrieve exact data**
- Works with **structured data**
- Returns **precise answers** to exact queries

✅ Example: Bank systems, student records, inventory databases

Matching Philosophy

- **IR System: Best Match**
  - Uses similarity measures (e.g., cosine similarity, BM25)
  - Results are **probabilistic**
  - "This document is *more relevant* than that one"
- **Database System: Exact Match**
  - Uses Boolean logic (AND, OR, NOT)
  - Results are **deterministic**
  - "This record satisfies the condition or it does not" (SELECT * FROM books WHERE topic = 'Information Retrieva

Web search



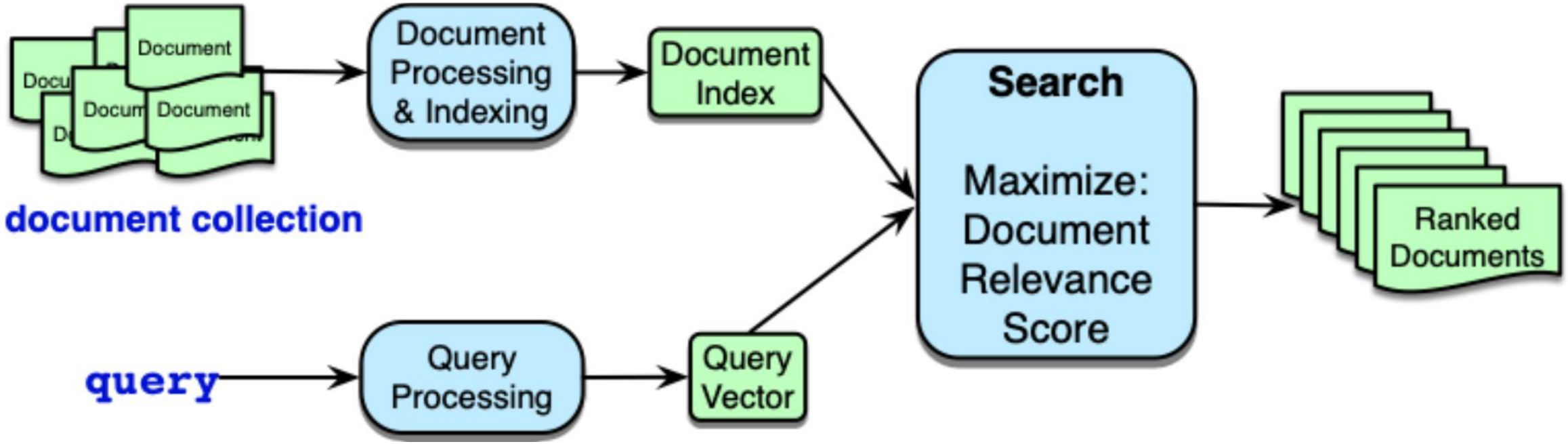
Not just the web

Major Uses of Information Retrieval

- Searching our email
- Searching corporate documents
- Searching personal medical records
- IR as a Core Component of Large Language Models (LLMs)
  - Retrieval-Augmented generation (RAG)
  - RAG combines:
    - Information Retrieval → find relevant documents
    - Language Models → generate answers using retrieved content

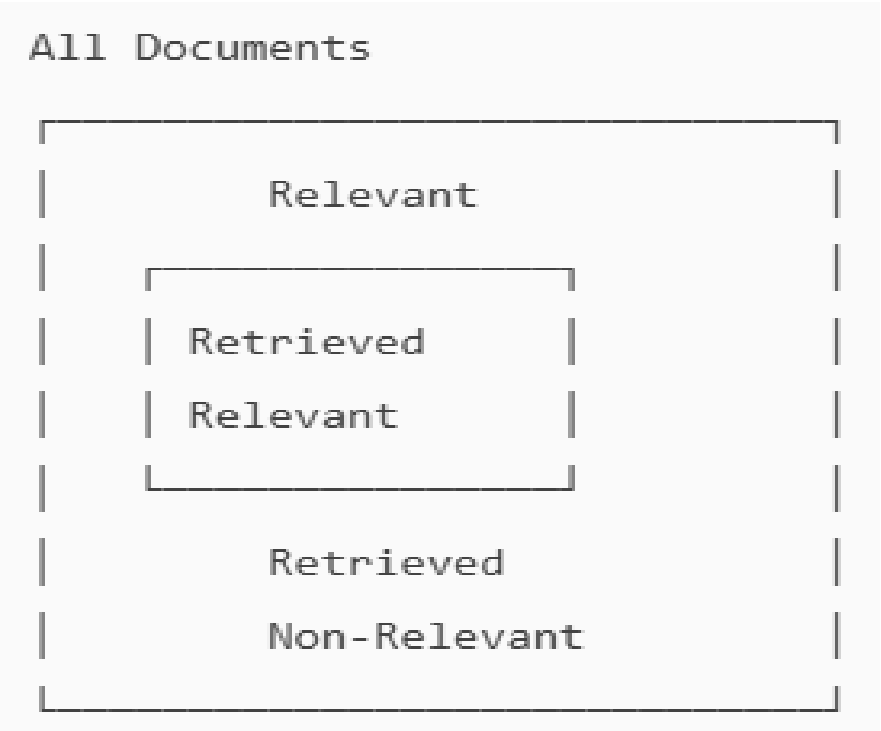
In most cases we do "ranked retrieval“ in IR

- The retriever returns top-k documents
- These are ranked
- We can show the user these, or some subset.



Document Relevance Score

- Goal is to assign **a score** to each document for whether it meets the user's information need
- We get **approximate scores** by the textual similarity between the query and the documents. **Relevant documents are ranked by their relevance score**.



Relevance: How well a document satisfies the user's information need

Precision: Fraction of retrieved documents that are relevant

Recall: Fraction of relevant documents that are retrieved

Two architectures

- Modern Information Retrieval systems mainly use **two architectures**:
  - **Sparse retrieval**
  - **Dense retrieval**
- Both represent **queries and documents as vectors** and compute **similarity** between them.

- **Sparse retrieval**
  - represent query and doc as vectors of word counts
  - weighted by tf-idf, BM25
- **Dense retrieval**
  - Use LLM to represent query and doc as embeddings

In both cases, similarity is **dot product** or **cosine similarity** between query and document representations

Sparse retrieval: The vector model of IR

- Gerard Salton (1971)introduced the **Vector Space Model (VSM)** of information retrieval
- In **1971**, Salton formalized the **Vector Space Model**, which established the core idea that:
  - **Queries and documents can be represented as vectors, and relevance can be computed using vector similarity.**
- This is exactly the idea underpinning **both sparse and dense retrieval today**.

Sparse Retrieval Representation

- Represent a document as a vector of counts of the words it contains.
- Query and documents are represented as high-dimensional sparse vectors
- Each dimension corresponds to a term (word) in the vocabulary
- Most values are zero

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth	...
Antony	1	1	0	0	0	1	
Brutus	1	1	0	1	0	0	
Caesar	1	1	0	1	1	1	
Calpurnia	0	1	0	0	0	0	
Cleopatra	1	0	0	0	0	0	
mercy	1	0	1	1	1	1	
worser	1	0	1	1	1	0	
...							

► **Figure 1.1** A term-document incidence matrix. Matrix element (t, d) is 1 if the play in column d contains the word in row t, and is 0 otherwise.

Bag-of-words model

- A document is represented by **the words it contains and their frequencies**, **ignoring grammar, word order, and syntax**.
  - Only **which words appear** and **how often** matters
  - The document is treated as a *"bag"* of words

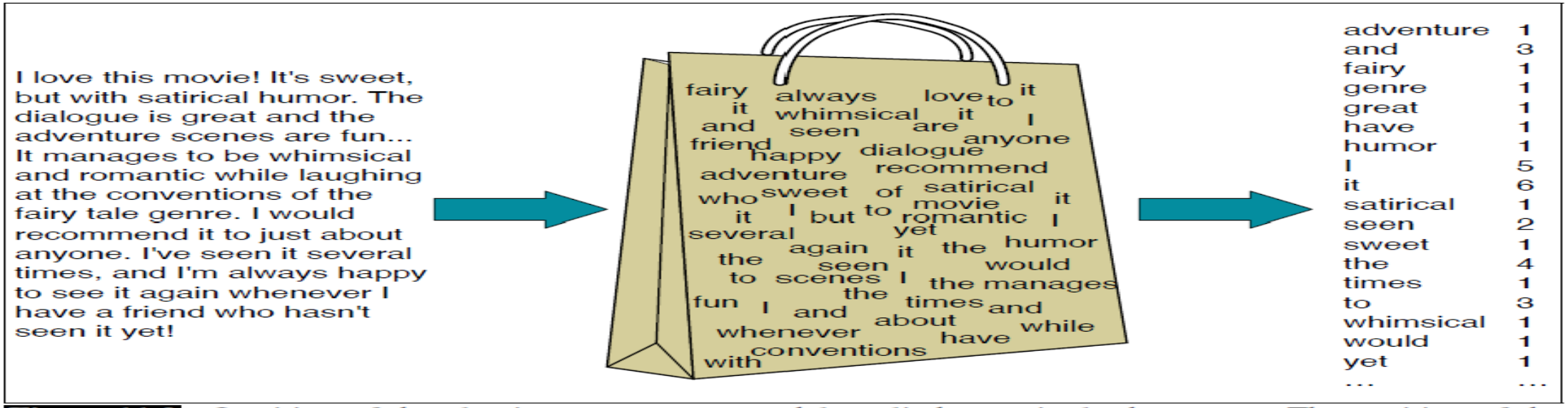


Figure 11.2 Intuition of the classic vector space model applied to a single document. The position of the words is ignored (the *bag-of-words* assumption) and we make use of the frequency of each word.

- Vector representation of the document : [1 3 1 1 1 1 1 5 6 1 2 1 4 1 3 1 1 1]
- Assuming that we limited ourselves to these 18 dimensions andignored all the other words in English

Term-document matrix

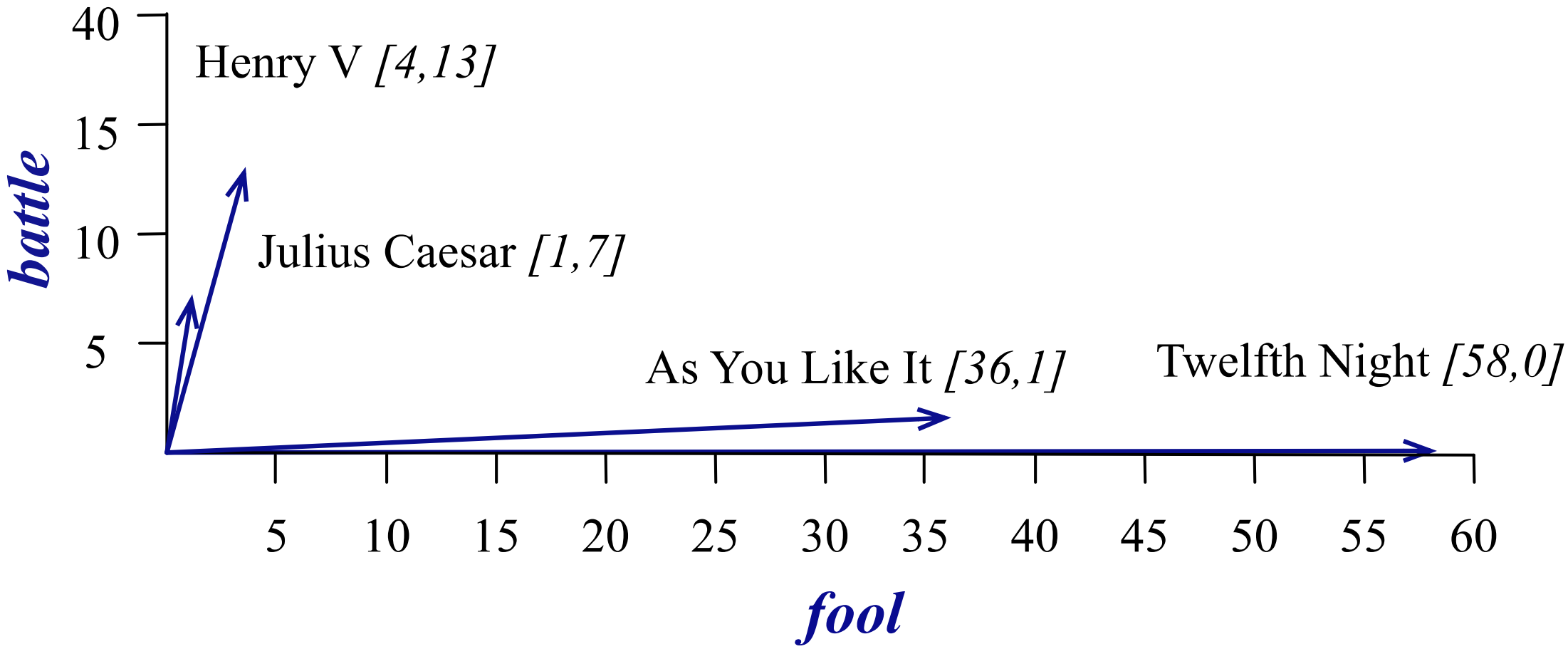
- Each document is represented by **a vector of words**
- The **term-document matrix** for four words in four Shakespeare plays. The **red boxes** show that each document is represented as a column vector of length four.

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

The term-document matrix for four words in four Shakespeare plays.

Visualizing document vectors

A spatial visualization of the document vectors for the four Shakespeare play documents, showing just two of the dimensions, corresponding to the words **battle** and **fool**. The comedies have high values for the **fool** dimension and low values for the **battle** dimension.



Vectors are the basis of information retrieval

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

- Vectors are similar for the two comedies
  - You Like It [1,114,36,20] and Twelfth Night [0,80,58,15] look a lot more like each other

- But comedies are different than the other two plays

  Comedies have more *fools* and *wit* and fewer *battles*.

Vector representations of queries and documents

- Suppose we are looking for a witty fool play:
- `Query = "fool wit"`

	As You Like It	Twelfth Night	Julius Caesar	Henry V	Query
battle	1	0	7	13	0
good	114	80	62	89	0
fool	36	58	1	4	1
wit	20	15	2	3	1

Choose the document that is most similar to the query

- Which of $\mathbf{d_1}$, $\mathbf{d_2}$, $\mathbf{d_3}$, $\mathbf{d_4}$ is most similar to **q**?

	d_1	d_2	d_3	d_4	q
	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**	**Query**
battle	1	0	7	13	0
good	114	80	62	89	0
fool	36	58	1	4	1
wit	20	15	2	3	1

Similarity methods are variants of dot product

- The dot product is **q · d**
- **score (q,d_1) = q · d_1 =**

	d_1	d_2	d_3	d_4	q
	As You Like It	**Twelfth Night**	**Julius Caesar**	**Henry V**	**Query**
battle	1	0	7	13	0
good	114	80	62	89	0
fool	36	58	1	4	1
wit	20	15	2	3	1

In fact we use cosine

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|} \cdot \frac{\mathbf{d}}{|\mathbf{d}|}$$

But raw frequency is a bad representation

- The co-occurrence matrices we have seen represent each cell by word frequencies.
- Frequency is clearly useful; if *sugar* appears a lot near *apricot*, that's useful information.
- But overly frequent words like *the*, *it,* or *they* are not very informative about the context
- It's a paradox! How can we balance these two conflicting constraints?

TF-IDF, PMI and BM25

TF-IDF (Term Frequency – Inverse Document Frequency)

- **TF-IDF** is a **term-weighting scheme** used in **Information Retrieval** to measure how important a word is to a document **relative to a collection**.
- It improves over raw word counts by:
  - rewarding **terms frequent in a document**
  - penalizing **terms common across many documents**

PMI — Pointwise Mutual Information

- **PMI (Pointwise Mutual Information)** is a statistical measure used to quantify **how strongly two events (typically words) are associated** compared to what we would expect if they were independent.
- In IR and NLP, PMI is often used to measure:
  - **Word–word association**
  - **Collocations** (e.g., *"information retrieval"*)
  - **Semantic relatedness** in sparse models

BM25 (Best Matching 25) is a probabilistic ranking function used in sparse information retrieval to score how relevant a document is to a query**. BM25** improves on **TF-IDF** by:

- Saturating term frequency
- Normalizing document length
- Using a probabilistic motivation

Two common solutions for word weighting

- **tf-idf:** tf-idf value for word t in document d:

$$w_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

Words like "the" or "it" have very low idf

- **PMI:** (Pointwise mutual information)
  - $\mathbf{PMI}(\boldsymbol{w_1}, \boldsymbol{w_2}) = \boldsymbol{log}\frac{\boldsymbol{p(w_1,w_2)}}{\boldsymbol{p(w_1)p(w_2)}}$

See if words like "good" appear more often with "great" than we would expect by chance

Term frequency (tf) in the tf-idf algorithm

- We could imagine using raw count:
- $$\mathrm{tf}_{t,d} = \mathrm{count}(t,d)$$
- But instead of using raw count, we usually squash a bit:

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

Document frequency (df)

- df_t *is* the number of documents *t* occurs in.
- (note this is not collection frequency: total count across all documents)
- "*Romeo*" is very distinctive for one Shakespeare play:

	Collection Frequency	Document Frequency
Romeo	113	1
action	113	31

Inverse document frequency (idf)

$$\text{idf}_t = \log_{10}\left(\frac{N}{\text{df}_t}\right)$$

N is the total number of documents in the collection

Word	df	idf
Romeo	1	1.57
salad	2	1.27
Falstaff	4	0.967
forest	12	0.489
battle	21	0.246
wit	34	0.037
fool	36	0.012
good	37	0
sweet	37	0

What is a document?

- Could be a play or a Wikipedia article
- But for the purposes of tf-idf, documents can be **anything**; we often call each paragraph a document!

Final tf-idf weighted value for a word

- Raw counts:

$$w_{t,d} = \text{tf}_{t,d} \times \text{idf}_t$$

- t

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	1	0	7	13
good	114	80	62	89
fool	36	58	1	4
wit	20	15	2	3

	As You Like It	Twelfth Night	Julius Caesar	Henry V
battle	0.246	0	0.454	0.520
good	0	0	0	0
fool	0.030	0.033	0.0012	0.0019
wit	0.085	0.081	0.048	0.054

TF-IDF: a worked example

An example of a tiny query against a collection of 4 nano documents

Query:	sweet love
Doc 1:	Sweet sweet nurse! Love?
Doc 2:	Sweet sorrow
Doc 3:	How sweet is love?
Doc 4:	Nurse!

TF-IDF weighted cosine

$$\text{score}(q,d) = \cos(\mathbf{q},\mathbf{d}) = \frac{\mathbf{q}}{|\mathbf{q}|} \cdot \frac{\mathbf{d}}{|\mathbf{d}|}$$

$$\text{score}(q,d) = \sum_{t \in \mathbf{q}} \frac{\text{tf-idf}(t,q)}{\sqrt{\sum_{q_i \in q} \text{tf-idf}^2(q_i,q)}} \cdot \frac{\text{tf-idf}(t,d)}{\sqrt{\sum_{d_i \in d} \text{tf-idf}^2(d_i,d)}}$$

TF-IDF weighted cosine

$$\mathrm{tf}_{t,d} = \begin{cases} 1 + \log_{10} \mathrm{count}(t,d) & \text{if } \mathrm{count}(t,d) > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\mathrm{idf}_t = \log_{10}\left(\frac{N}{\mathrm{df}_t}\right)$$

$$\mathrm{score}(q,d) = \sum_{t \in \mathbf{q}} \frac{\text{tf-idf}(t,q)}{\sqrt{\sum_{q_i \in q} \text{tf-idf}^2(q_i,q)}} \cdot \frac{\text{tf-idf}(t,d)}{\sqrt{\sum_{d_i \in d} \text{tf-idf}^2(d_i,d)}}$$

TF-IDF nano-example

Query: sweet love

Doc 1: Sweet sweet nurse! Love?

Doc 2: Sweet sorrow

Doc 3: How sweet is love?

Doc 4: Nurse!

TF-IDF nano-example

Query						
word	**cnt**	**tf**	**df**	**idf**	**tf-idf**	**n'lized** = tf-idf/$\|q\|$
sweet	1	1	3	0.125	0.125	0.383
nurse	0	0	2	0.301	0	0
love	1	1	2	0.301	0.301	0.924
how	0	0	1	0.602	0	0
sorrow	0	0	1	0.602	0	0
is	0	0	1	0.602	0	0
$\|q\| = \sqrt{.125^2 + .301^2} = .326$						

TF-IDF nano-example

Computation of tf-idf cosine score between the query and nano-documents 1 and 2

	Document 1					Document 2				
word	**cnt**	**tf**	**tf-idf**	**n'lized**	**× q**	**cnt**	**tf**	**tf-idf**	**n'lized**	**×q**
sweet	2	1.301	0.163	0.357	**0.137**	1	1.000	0.125	0.203	**0.0779**
nurse	1	1.000	0.301	0.661	0	0	0	0	0	0
love	1	1.000	0.301	0.661	**0.610**	0	0	0	0	**0**
how	0	0	0	0	0	0	0	0	0	0
sorrow	0	0	0	0	0	1	1.000	0.602	0.979	0
is	0	0	0	0	0	0	0	0	0	0
$\lvert d_1 \rvert = \sqrt{.163^2 + .301^2 + .301^2} = .456$						$\lvert d_2 \rvert = \sqrt{.125^2 + .602^2} = .615$				
Cosine: $\sum$ of column: **0.747**						Cosine: $\sum$ of column: **0.0779**				

Goal: rank documents in D by their TF-IDF-weighted cosines with query q

Do we have to consider all documents in D?

No!

We can ignore all documents that don't have any query words!

They will have a cosine of 0!

Efficiency with The Inverted Index

How do we efficiently find all documents that contain a query term $\mathbf{q_i}$?

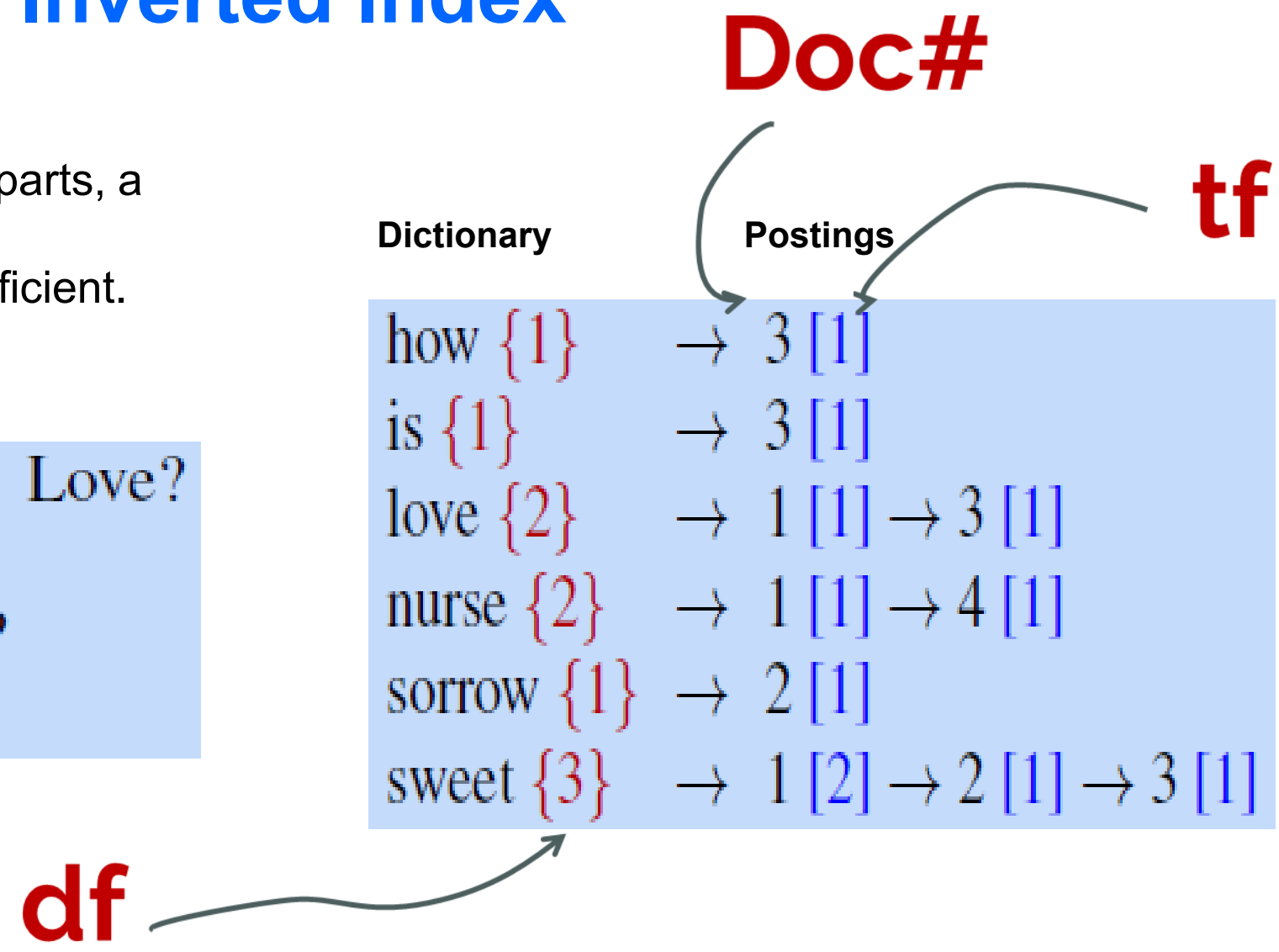
Use an index!

Which for historical reasons we call **an inverted index**!

Inverted Index

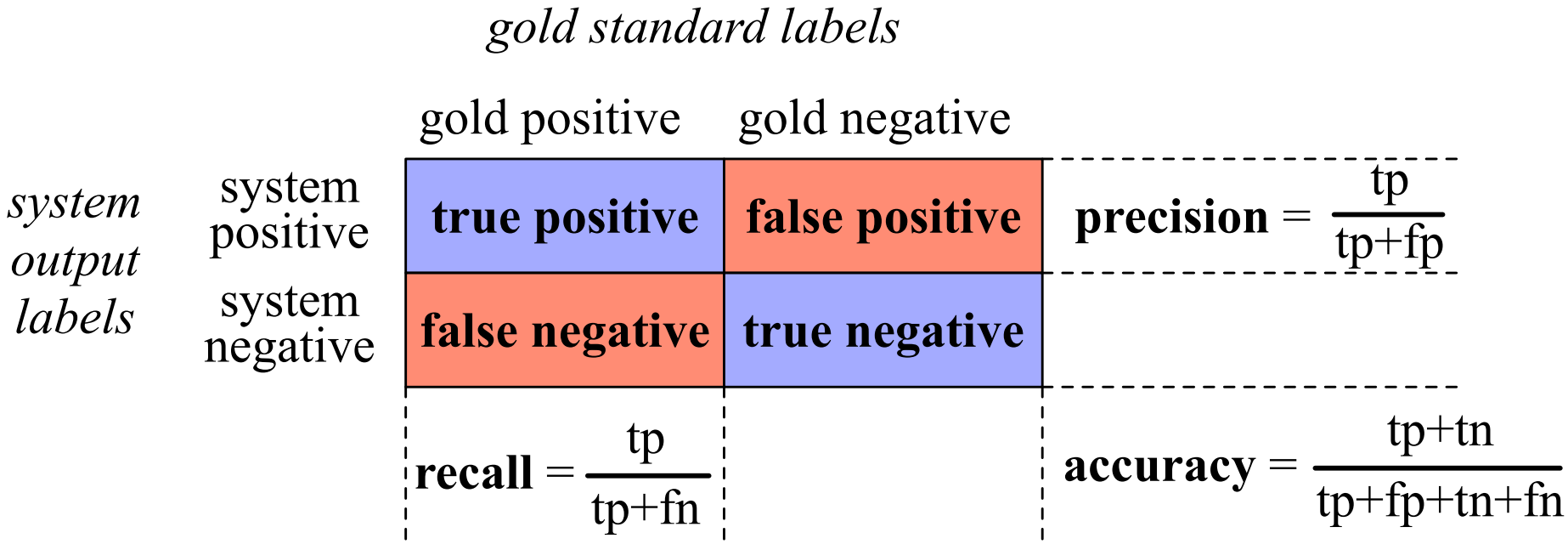
- An inverted index consists of two parts, a dictionary and the postings.
- we use it for making the search efficient.

Doc 1: Sweet sweet nurse! Love?
Doc 2: Sweet sorrow
Doc 3: How sweet is love?
Doc 4: Nurse!



Evaluation of IR: Precision and Recall

We saw these already for classification



Precision: % of selected items that are correct

Recall: % of correct items that are selected

Precision and Recall for IR

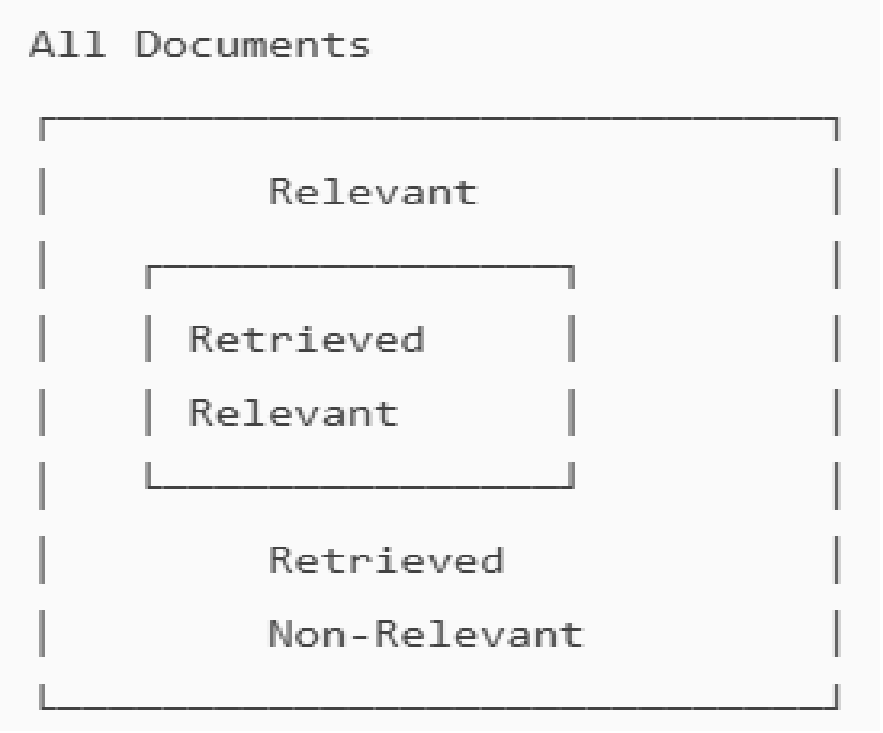
User makes an information request

Every document in collection is either:

- **Relevant** to the user
- **Not relevant** to the user

•The system retrieves a ranked set of documents

Evaluation of IR



Relevance: How well a document satisfies the user's information need

Precision: Fraction of retrieved documents that are relevant

Recall: Fraction of relevant documents that are retrieved

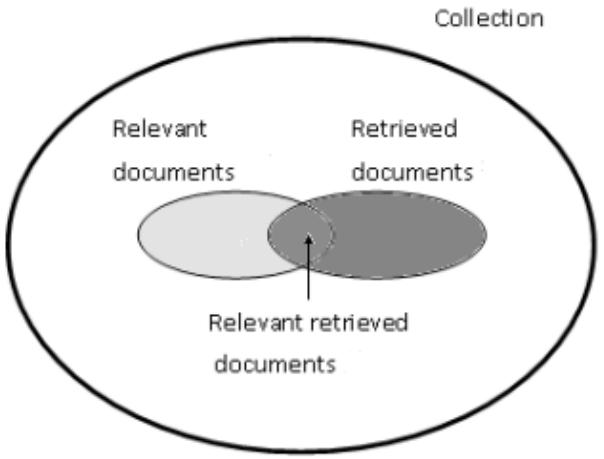
Precision for IR

Precision = % of **retrieved** documents that are **relevant**

System retrieves two kinds of documents

relevant documents

irrelevant documents

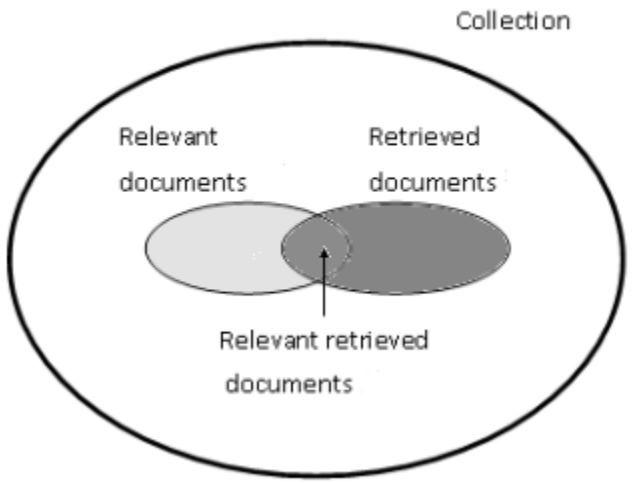


$$\textbf{Precision} = \frac{|\textbf{relevant retrieved} \text{ docs}|}{|\textbf{relevant retrieved} \text{ docs}| + |\textbf{irrelevant retrieved} \text{ docs}|}$$

Recall for IR

Recall = % of **relevant** documents that are **retrieved**

$$\textbf{Recall} = \frac{|\text{retrieved } \textbf{relevant} \text{ documents}|}{|\text{all } \textbf{relevant} \text{ document}|}$$



Problem with classic IR

- The **vocabulary mismatch problem**
- tf-idf or BM25 cosine similarities only work if there is **exact word overlap** between query and doc!
- But query-writer can't know the exact words the doc might include!

Dense Retrieval

Representation

- Query and documents are represented as **dense vectors (embeddings)**
- Generated using **neural models / LLMs**
- Lower dimensional (e.g., 384, 768, 1024)

Ex:

```
Query embedding:
[0.12, -0.34, 0.88, ..., 0.05]
Document embedding:
[0.10, -0.30, 0.91, ..., 0.07]
```

Dense Retrieval

Sparse Retrieval vs Dense Retrieval:
Sparse retrieval represents text using weighted word counts (TF-IDF, BM25), while dense retrieval uses neural embeddings; in both cases, relevance is computed via vector similarity such as dot product or cosine similarity.

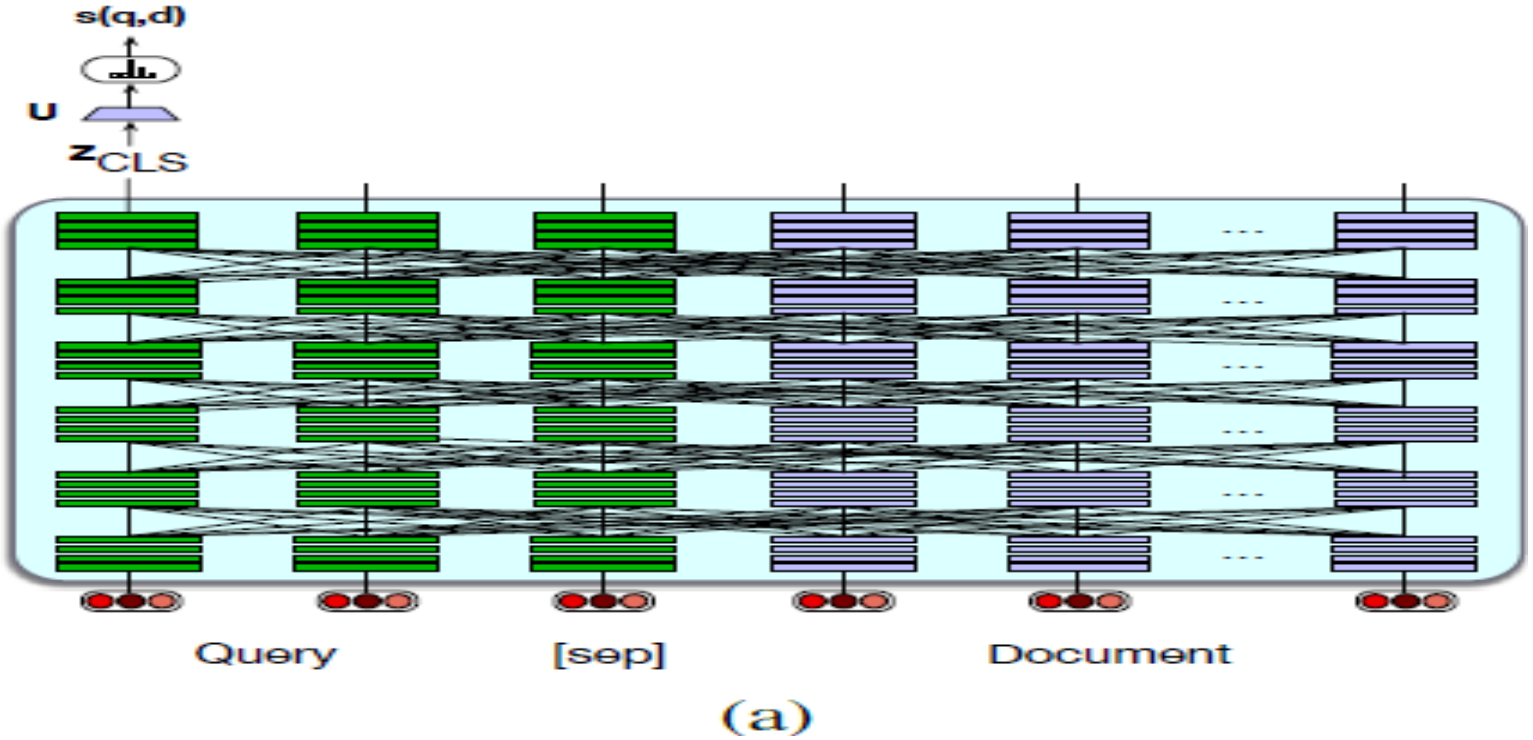
Aspect	Sparse Retrieval	Dense Retrieval
Representation	Word counts	Embeddings
Model	TF-IDF, BM25	LLM / neural encoder
Vocabulary	Explicit terms	Latent semantics
Vector size	Very large, sparse	Small, dense
Semantic matching	Weak	Strong
Typical use	Search engines	RAG, semantic search

Hypothetical version of dense retrieval: static embeddings

- Sparse representation work only if there is exact overlap of words between the query and document.
- The solution to this problem is to use an approach that can handle synonymy:
  - instead of (sparse) word-count vectors, using (dense) embeddings.
  - Replace tf-idf vectors with, e.g., word2vec
    - Query: the mean of the embeddings of each query word
    - Doc: the mean of the doc word embeddings
    - Now just compute query-doc cosine as normal.
- We don't do this
- because **contextual embeddings** work much better!

Dense retrieval #1: Single encoder

- The most powerful approach is to present both **the query** and **the document** to a single encoder, allowing the **transformer self-attention** to see all the tokens of both the query and the document, and thus building a representation that is sensitive to the meanings of both query and document.



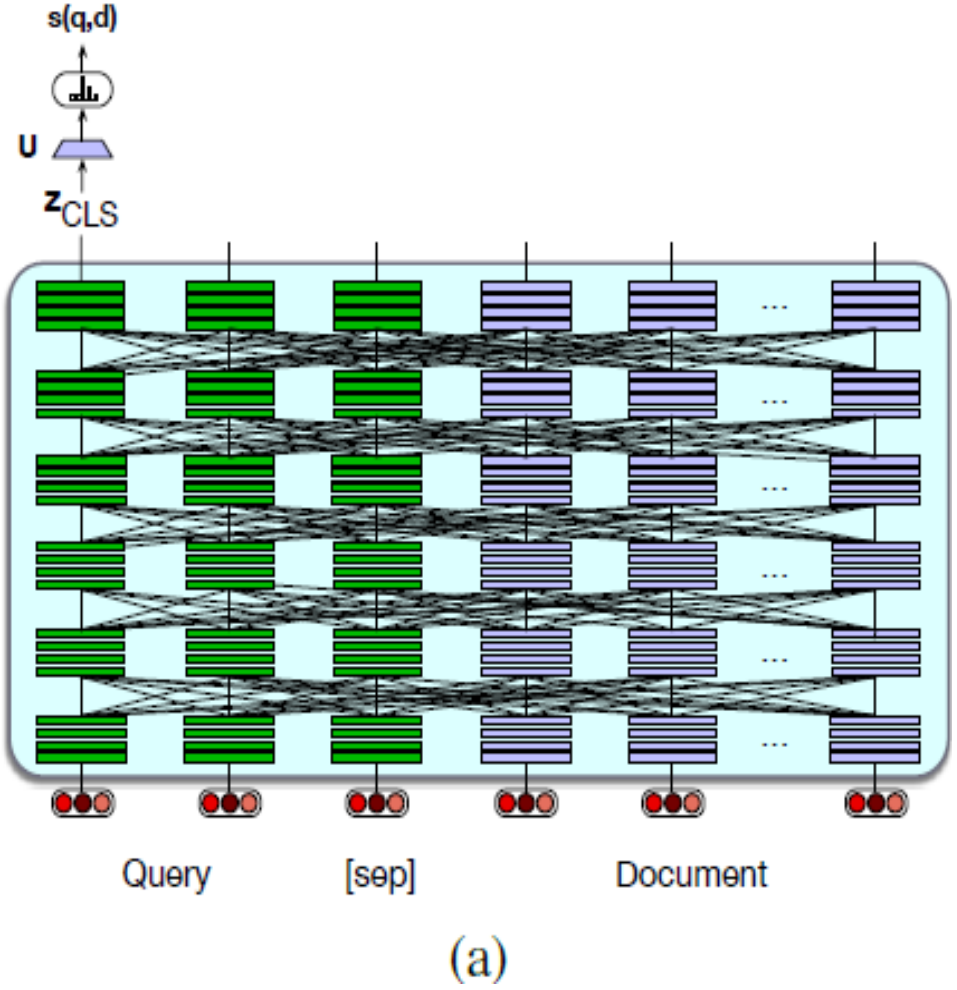
$$\mathbf{z} = \text{BERT}(q; [\text{SEP}]; d)[\text{CLS}]$$

$$\text{score}(q,d) = \text{softmax}(\mathbf{U}(\mathbf{z}))$$

The BERT system together with the linear layer U can then be fine-tuned for the relevance task by gathering a tuning dataset of relevant and non-relevant passages.

- Use a single encoder to jointly encode query and document and finetune to produce a relevance score with a linear layer over the CLS token. This is too compute-expensive to use except in rescoring

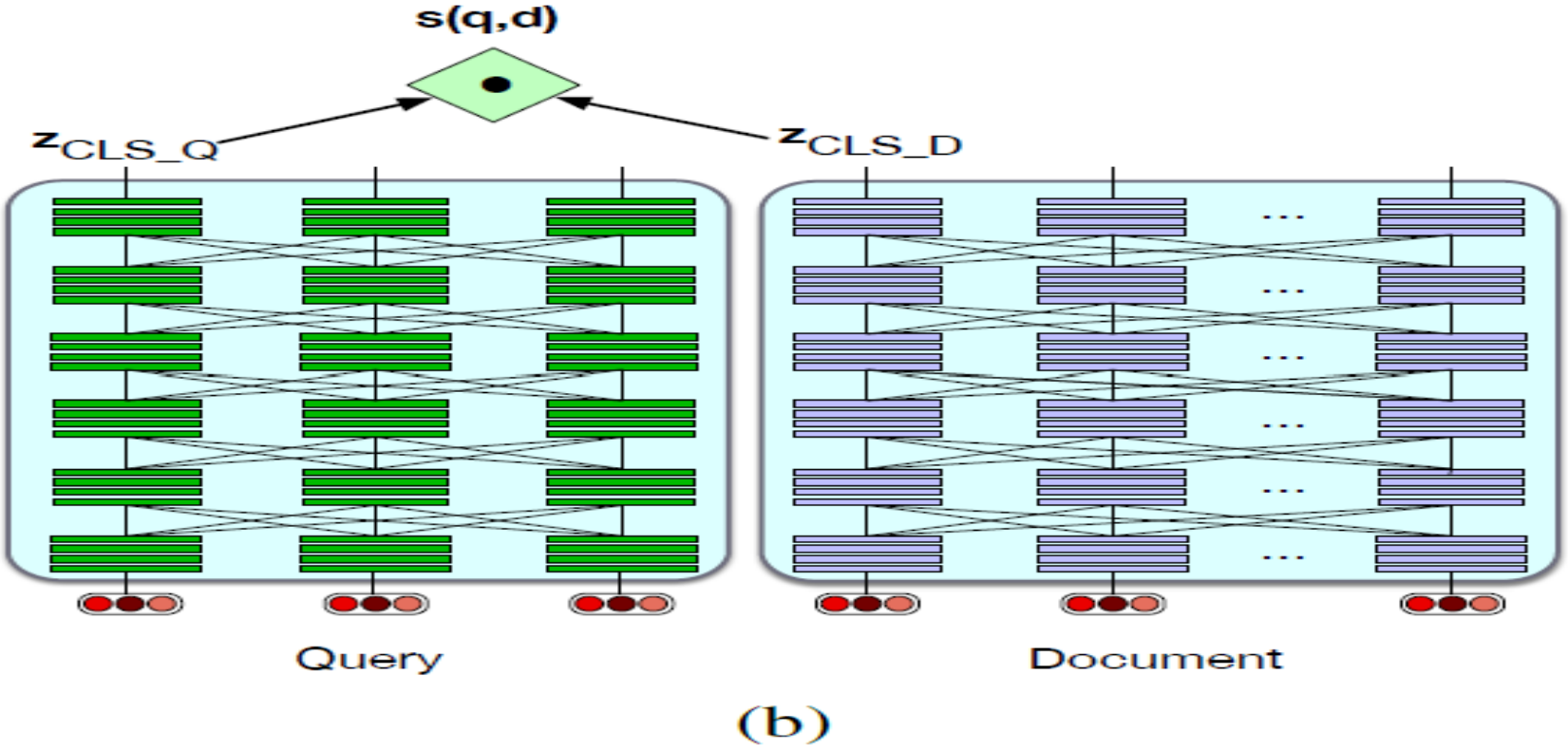
Dense retrieval #1: Single encoder



- Usually the retrieval step is not done on an entire document.
- Instead documents are broken up into smaller passages, such as non-overlapping fixed-length chunks of say 100 tokens, and the retriever encodes and retrieves these passages rather than entire documents.
- **Training:**
- BERT and linear layer U can then fine-tuned for relevance
- Creating a tuning dataset of relevant and non-relevant passages.

Dense retrieval #2 (bi-encoder)

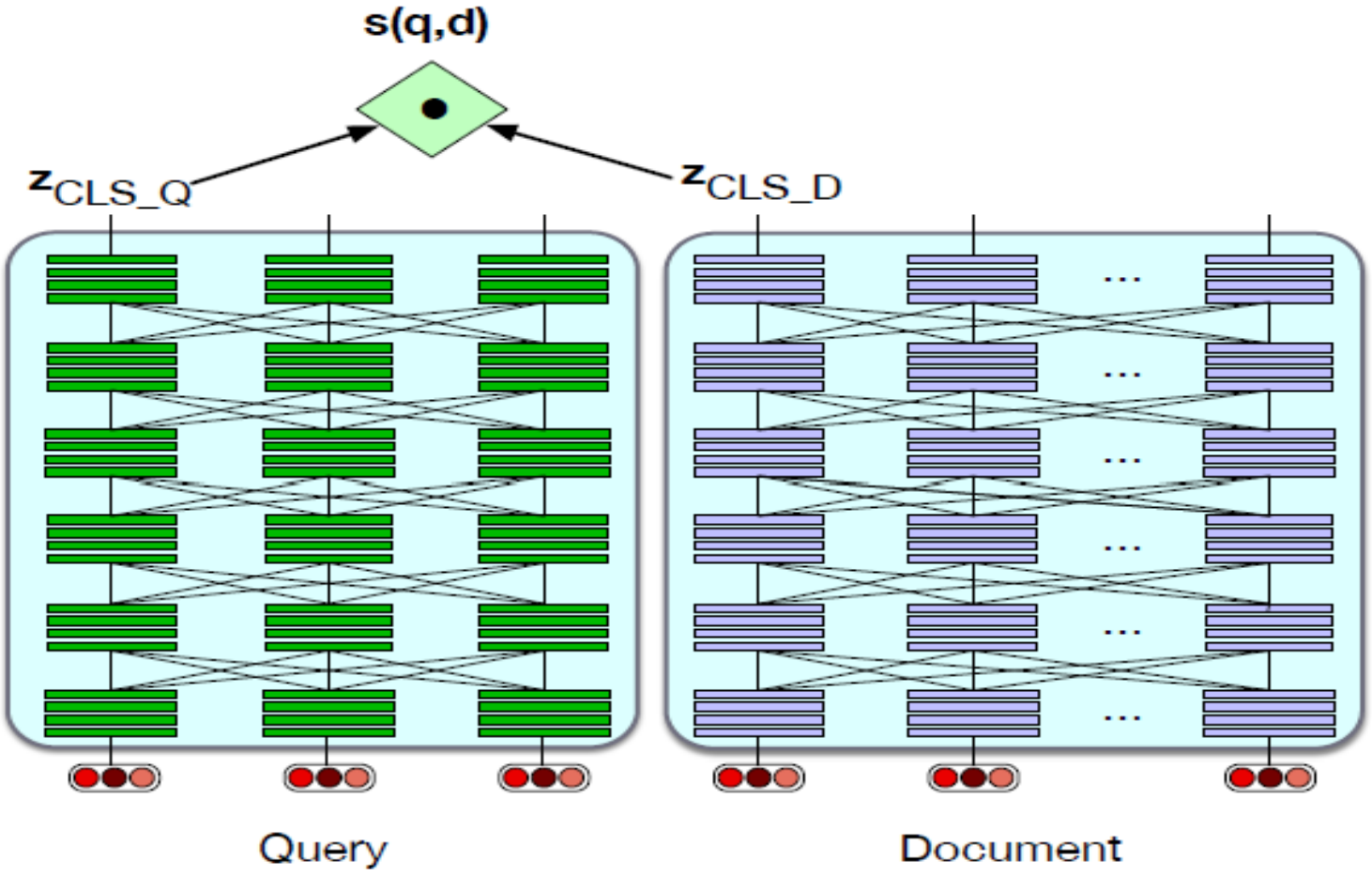
- At the other end of the computational spectrum is **a much more efficient architecture**, the bi-encoder. In this architecture we can encode the documents in the collection only one time by using two separate encoder models, one to encode the query and one to encode the document.



$$\mathbf{z}_q = \text{BERT}_Q(\text{q})[\text{CLS}]$$
$$\mathbf{z}_d = \text{BERT}_D(\text{d})[\text{CLS}]$$
$$\text{score}(q,d) = \mathbf{z}_q \cdot \mathbf{z}_d$$

Use separate encoders for query and document, and use the dot product between CLS token outputs for the query and document as the score. This is less compute-expensive, but not as accurate.

Dense retrieval #2 (biencoder)



(b)

- Encode doc vectors in advance.
- Encode query when it arrives
- Score is dot product between query vector and precomputed doc vector
- Cheaper but less accurate

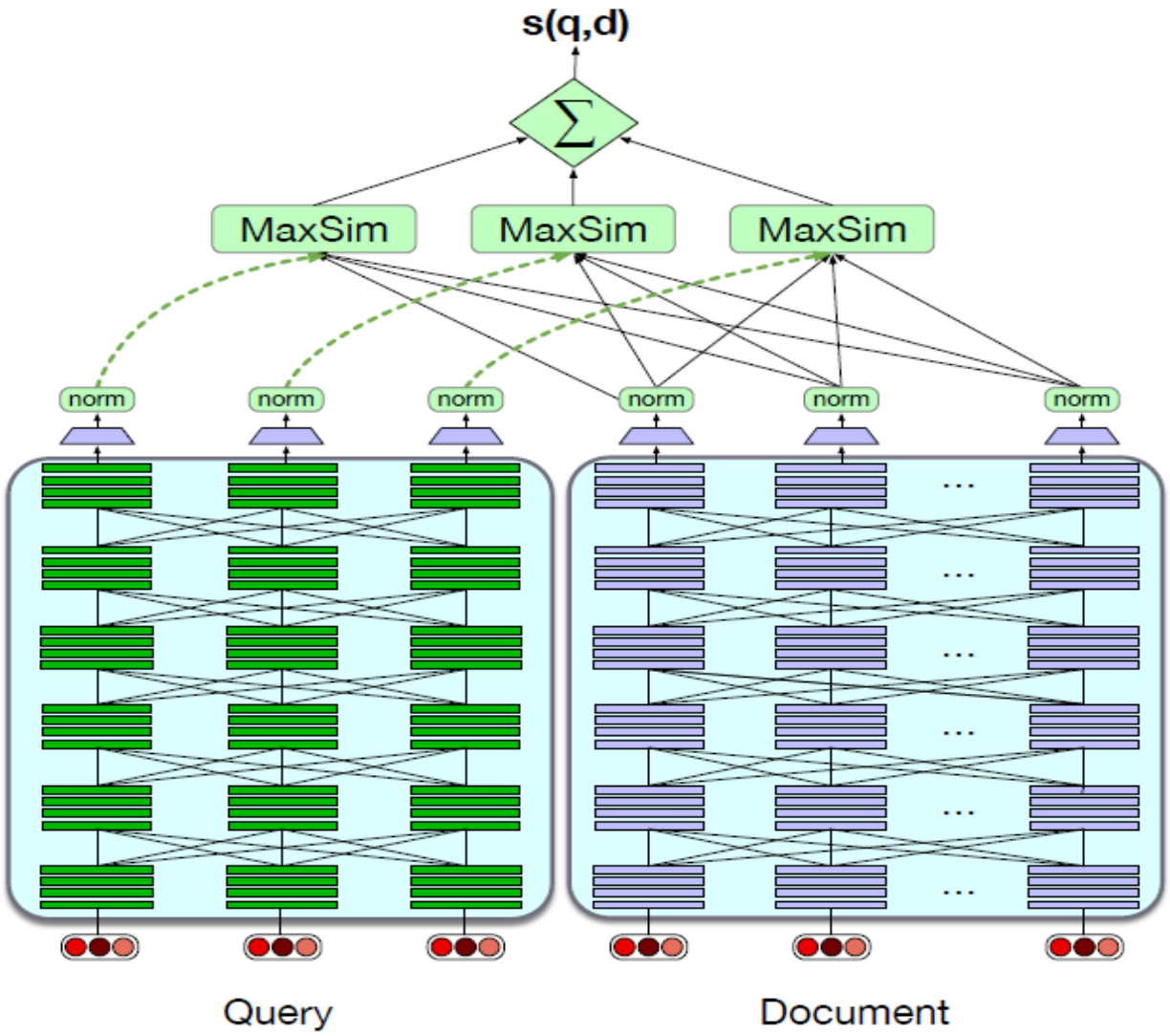
In-between dense retrieval methods

- Use cheap methods (like BM25) as first pass relevance ranking for each document,
  - Then just rerank the top N ranked docs,
  - Using expensive methods like the full BERT scoring

ColBERT

- **Another intermediate approach is** the ColBERT approach of Khattab and Zaharia (2020) and Khattab et al. (2021), shown in Fig. 11.12.
- This method **separately encodes the query and document**, but rather than encoding the entire query or document into one vector, it separately encodes each of them into contextual representations for each token.
- These BERT representations of each document word can be **pre-stored for efficiency**.
- The **relevance score** between a query q and a document d is a sum of maximum similarity (MaxSim) operators between tokens in q and tokens in d.

ColBERT



$$\text{score}(q,d) = \sum_{i=1}^{N} \max_{j=1}^{m} \mathbf{E}_{q_i} \cdot \mathbf{E}_{d_j}$$

Training for dense retrieval

- ColBERT and other models need to be trained
  - To fine-tune the BERT encoders and train the linear layers (and the special [Q] and [D] embeddings)
  - On datasets of triples ⟨q,d+,d−⟩ triples of query q, positive document d+ and negative document d-to produce a score for each document
  - Some datasets like MS MARCO Ranking have positive examples

Efficiency in dense retrieval

- We must rank **every document** for its similarity to the query!
- **Efficiency for sparse word-count vectors:** inverted index
- **Efficiency for dense retrieval:**
  - For dense vector algorithms **finding the set of dense document vectors that have the highest dot product** with a dense query vector is an instance of the problem of nearest neighbor search.
  - **nearest neighbor search**: finding the set of dense document vectors that have the highest dot product with a dense query vector.
  - Modern systems make use of approximate nearest neighbor vector search algorithms like **Faiss** (Johnson et al., 2017).
    - Approximates the doc vector by a smaller quantized vector

Retrieval-Augmented Generation
RAG

- The information retrieval techniques we introduced in the prior section can be **integrated into language models** via a method called retrieval-augmented generation or RAG.
- In the basic RAG scenario that we will describe in this section, we use IR techniques to **retrieve documents** from some specified store of documents that are likely to have useful information.
- Then we use a large language model to **generate an answer conditioned on these documents** in addition to the original query.

IR plays a central role in modern LLMs

- How to answer factual questions like
- Where is the Louvre Museum located?
- How to get a script l in latex?
- Where does the energy in a nuclear explosion come from?

Just prompt an LLM!

AI Overview

The main Louvre Museum is located in **Paris, France**, **at the Musée du Louvre, 75001 Paris, France**. It is situated on the Right Bank of the Seine River in the city's 1st arrondissement, housed within the historic Louvre Palace.

- **Address:** Rue de Rivoli, 75001 Paris, France.

- LLMs seem to store facts in the connections in their feedforward layers!

But there are issues!

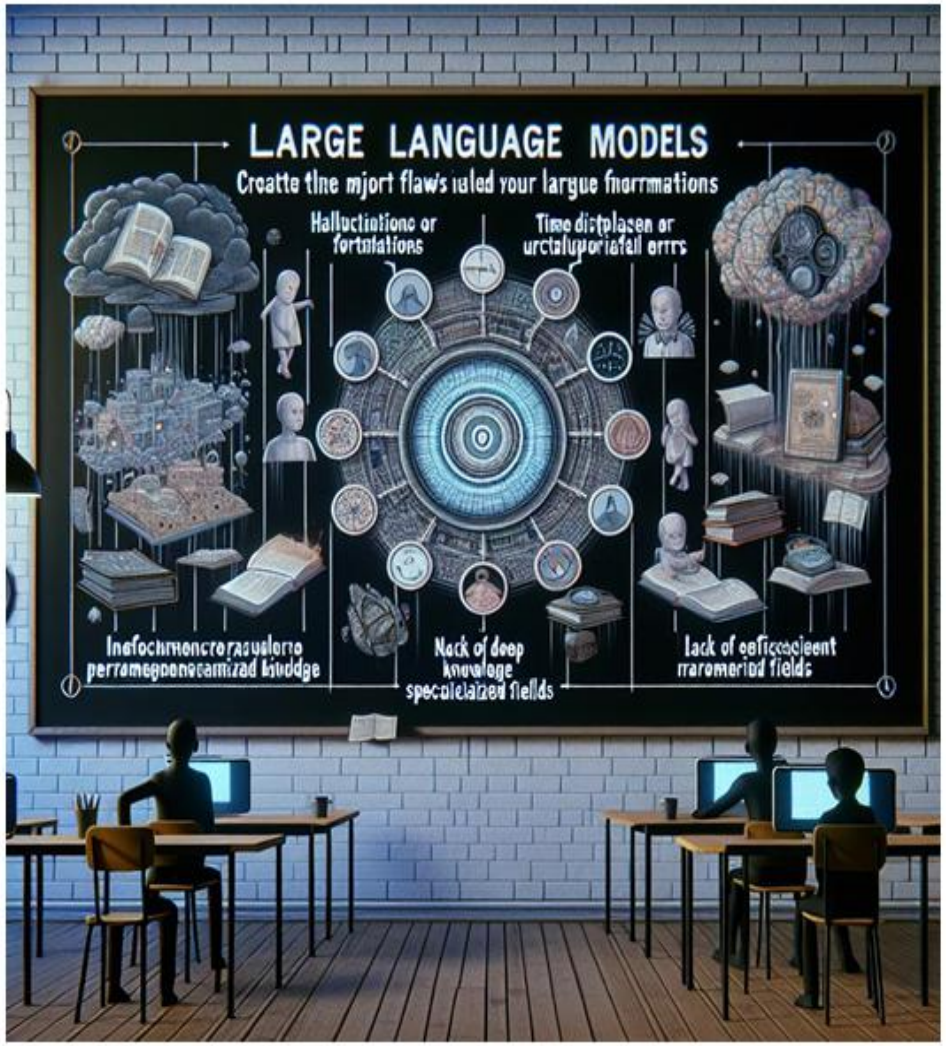
- LLMs Hallucinate
  - Hallucination: a response that is not faithful to the facts of the world.
  - In the legal domain LLMs were shown to hallucinate up to 88% of the time! Dahl et al. (2024)
- Can't use Proprietary Data
  - People need to ask questions about:
    - personal email.
    - healthcare applications to medical records.
    - internal corporate documents
    - legal documents discovery
- Can't Handle Dynamic Data
  - LLMs can't answer questions about rapidly changing information
  - Things that happened last week
  - In general, data shifts over time

Drawbacks of LLMs

- Hallucination
- Outdated information
- Low efficiency in parameterizing knowledge
- Lack of in-depth knowledge in specialized domains
- Weak inferential capabilities

Practical Requirements of Application

- Domain-specific accurate answering
- Frequent updates of data
- Traceability and explainability of generated content
- Controllable Cost
- Privacy protection of data



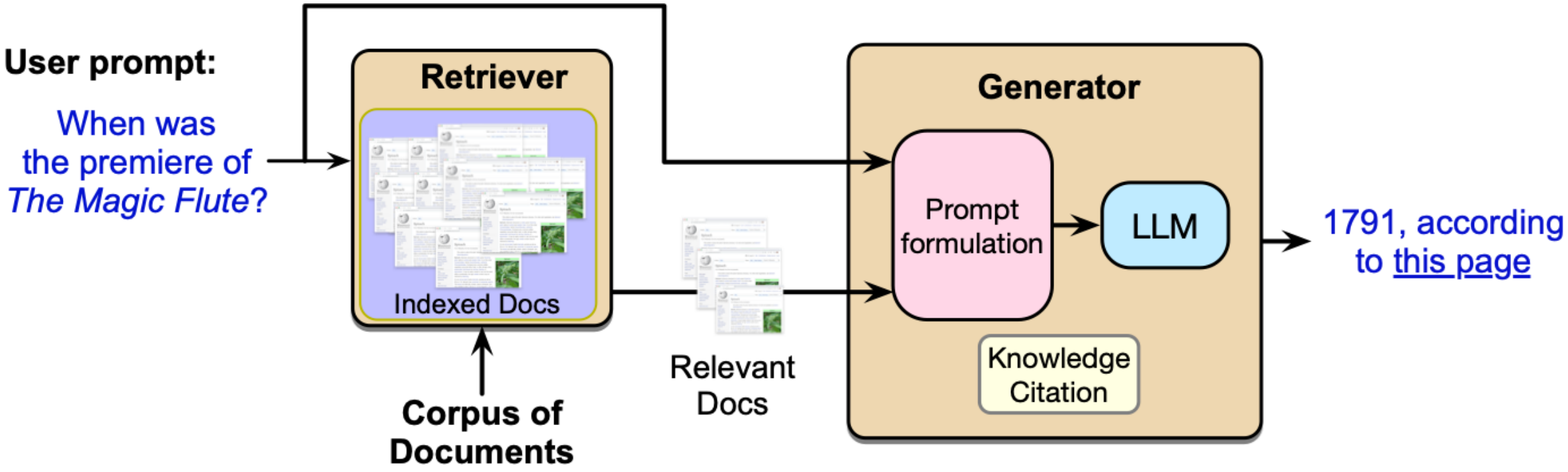
Draw by DALL·E-3

Source: Haofen Wang

Solution: RAG

- Retrieval-Augmented Generation
  1. Use IR to **retrieve** documents from some collection
  2. Then use LLM to **generate** an answer conditioned on the documents

Retrieval Augmented Generation (RAG)



Basic RAG

- Given a document collection D and a user query q
- Call a retriever to return top *k* passages
- Create a prompt that includes *q* and the passages
- Call an LLM with the prompt

Schematic of a RAG Prompt

```
retrieved passage 1

retrieved passage 2

...

retrieved passage k

Based on these texts, answer this question:  What year
was the premiere of The Magic Flute?
```

RAG

The task for the language model is then to generate text according to this probability model:

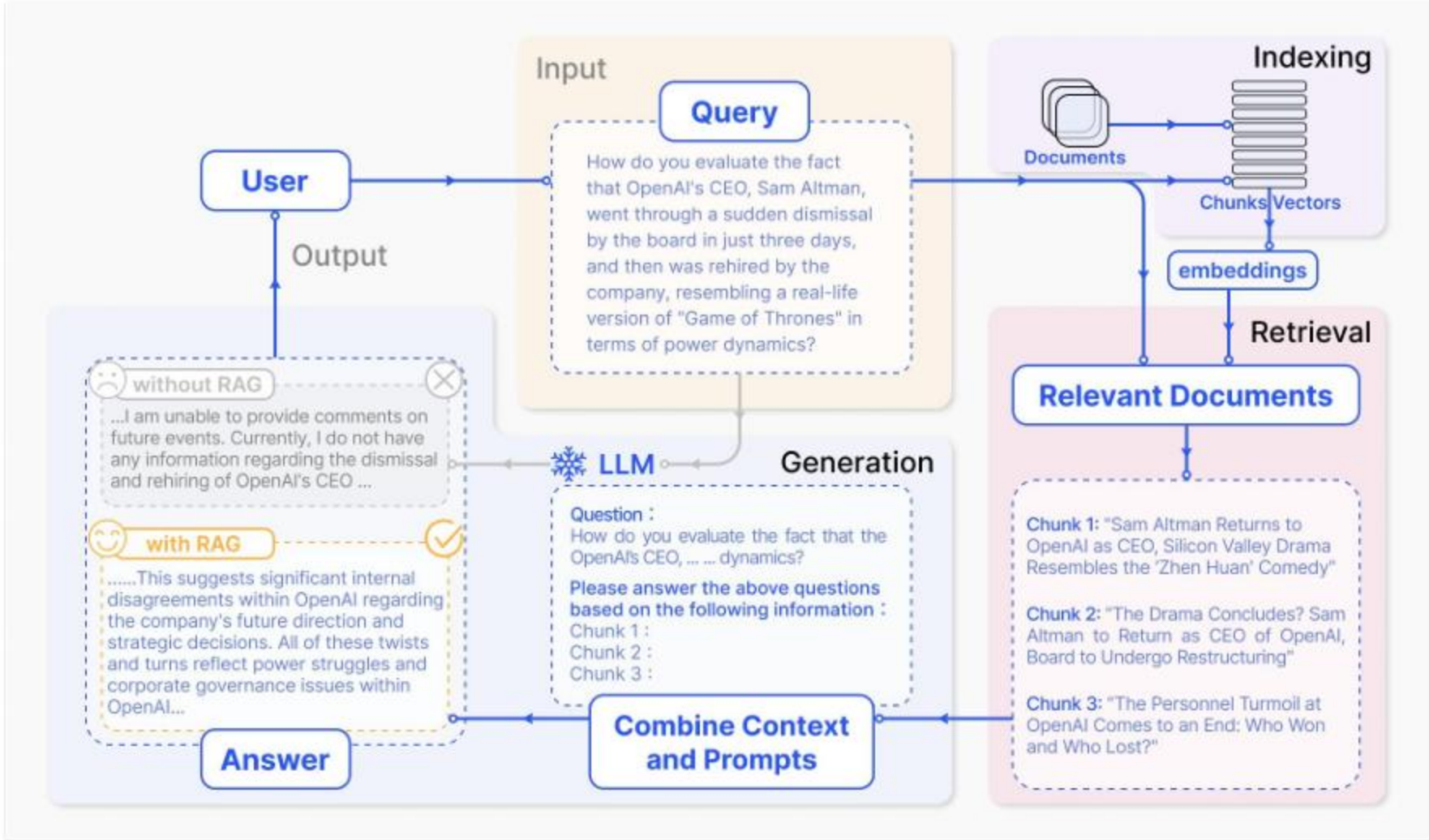
$$p(x_1, \ldots, x_n) = \prod_{i=1}^{n} p(x_i | \mathrm{R}(q)\,;\ \text{Answer the following question...}\ ;\ q\ ; x_{<i})$$

Retrieval-Augmented Generation (RAG)

When answering questions or generating text, it first retrieves relevant information from a large number of documents, and then LLMs generates answers based on this information.

By attaching a external knowledge base, there is no need to retrain the entire large model for each specific task.

The RAG model is especially suitable for knowledge-intensive tasks.



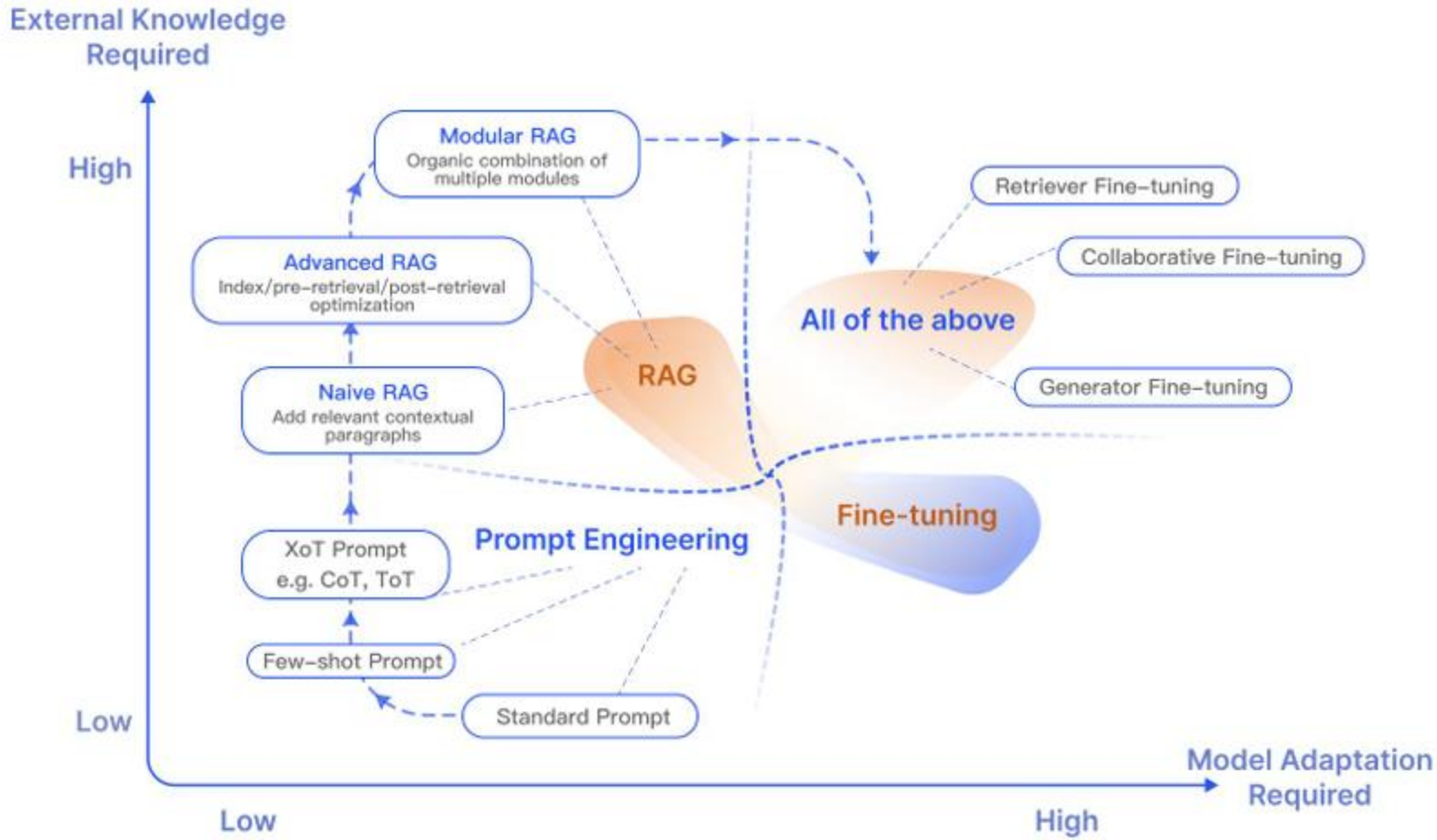
A typical case of RAG

Source: Haofen Wang

Symbolic Knowledge or Parametfic Knowledge

Ways to optimize LLMs.

- Prompt Engineering
- Retrieval-Augmented Generation
- Instruct / Fine-tuning



A typical case of RAG

Source: Haofen Wang

RAG vs Fine-tuning

Source: Haofen Wang

Feature Comparison	RAG	Fine-Tuning
Knowledge Updates	Directly updating the retrieval knowledge base ensures that the information remains current without the need for frequent retraining, making it well-suited for dynamic data environments.	Stores static data, requiring retraining for knowledge and data updates.
External Knowledge	Proficient in leveraging external resources, particularly suitable for accessing documents or other structured/unstructured databases.	Can be utilized to align the externally acquired knowledge from pretraining with large language models, but may be less practical for frequently changing data sources.
Data Processing	Involves minimal data processing and handling.	Depends on the creation of high-quality datasets, and limited datasets may not result in significant performance improvements.
Model Customization	Focuses on information retrieval and integrating external knowledge but may not fully customize model behavior or writing style.	Allows adjustments of LLM behavior, writing style, or specific domain knowledge based on specific tones or terms.
Interpretability	Responses can be traced back to specific data sources, providing higher interpretability and traceability.	Similar to a black box, it is not always clear why the model reacts a certain way, resulting in relatively lower interpretability.
Computational Resources	Depends on computational resources to support retrieval strategies and technologies related to databases. Additionally, it requires the maintenance of external data source integration and updates.	The preparation and curation of high-quality training datasets, defining fine-tuning objectives, and providing corresponding computational resources are necessary.
Latency Requirements	Involves data retrieval, which may lead to higher latency.	LLM after fine-tuning can respond without retrieval, resulting in lower latency.
Reducing Hallucinations	Inherently less prone to hallucinations as each answer is grounded in retrieved evidence.	Can help reduce hallucinations by training the model based on specific domain data but may still exhibit hallucinations when faced with unfamiliar input.
Ethical and Privacy Issues	Ethical and privacy concerns arise from the storage and retrieval of text from external databases.	Ethical and privacy concerns may arise due to sensitive content in the training data.

RAG Applications

Scenarios where RAG is applicable:

- Long-tail distribution of data
- Frequent knowledge updates
- Answers requiring verification and traceability
- Specialized domain knowledge
- Data privacy preservation

Q&A
RETRO (Borgeaud et al2021)
REALM (Gu et al, 2020)
ATLAS (Izacard et al, 2023)

Fact Checking
RAG (Lewis et al, 2020)
ATLAS (Izacard et al, 2022)
Evi. Generator (Asai et al, 2022a)

Dialog
BlenderBot3 (Shuster et al.2022)
Internet-augmented generation (Komeili et a., 2022)

Summary
FLARE (Jiang et al, 2023)

Machine Translation
kNN-MT (Khandelwal et al., 2020)TRIME-MT (Zhong et al., 2022)

Code Generation
DocPrompting (Zhou et al., 2023
Natural ProverWelleck et al., 2022)

Natural Language Inference
kNN-Prompt (Shi et al., 2022)
NPM (Min et al., 2023)

Sentiment analysis
kNN-Prompt (Shi et al., 2022)NPM (Min et al., 2023)

Commonsense reasoning
Raco (Yu et al, 2022)

Source: Haofen Wang

Naive RAG

Step1 Indexing

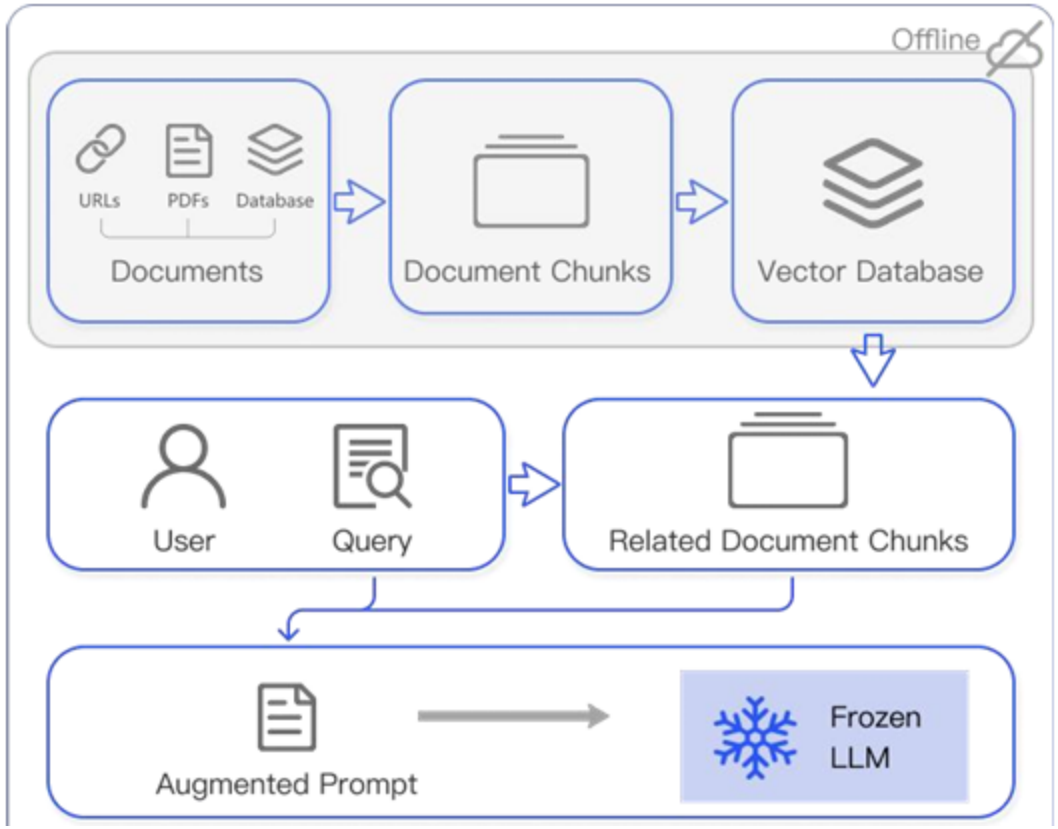
1. Divide the document into even chunks, each chunk being a piece of the original text.
2. Using the encoding model to generate an embedding for each chunck.
3. Store the Embedding of each block in the vector database.

Step2 Retrival

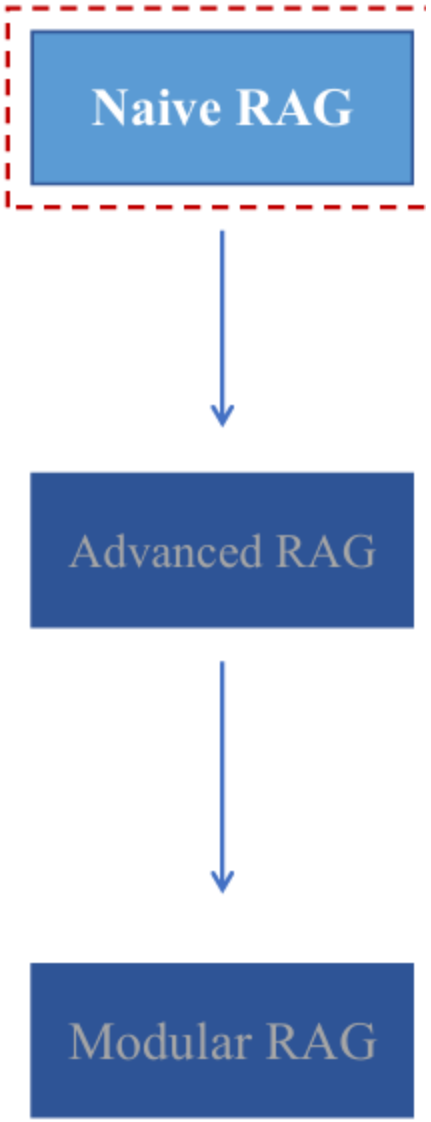
Retrieve the k most relevant documents using vector similarity search.

Step3 Generation

The original query and the retrieved text are combined and input into a LLM to get the final answer



Source: Haofen Wang



Chunks/Passages in RAG (Retrieval Augmented Generation)

- 1️⃣ **What is a "Chunk"?**
- In RAG, a **chunk** is a **small, retrievable unit of text** created by splitting a larger document (PDF, web page, book, codebase, database record).
- Instead of embedding or retrieving entire documents, RAG systems work with **chunks** so that:
  - Retrieval is more precise
  - Context windows are used efficiently
  - The LLM sees only relevant information
- **Simple definition**
- A chunk is the smallest meaningful piece of information you retrieve and send to the LLM.

Chunks/Passages in RAG (Retrieval Augmented Generation)

- **2️⃣ Why Chunking Is Critical in RAG**
- Chunking directly affects **retrieval quality**, which in turn affects **answer quality**.
- Poor chunking ⇒ perfect embeddings still fail.
- **Key reasons chunking matters**

Problem	**Without Proper Chunking**
Long documents	Relevant info buried
Semantic search	Embeddings too generic
Context window	Quickly exhausted
Hallucinations	Missing key facts
Cost	More tokens, higher latency

Chunks/Passages in RAG (Retrieval Augmented Generation)

- **3 The Chunking Pipeline**
- A typical RAG ingestion pipeline:
  - Raw Documents
    ↓
  - Text Cleaning
    ↓
  - Chunking (Split into pieces)
    ↓
  - Metadata Attachment
    ↓
  - Embedding Generation
    ↓
  - Vector Database
- Chunking is **done once during indexing**, not at query time.

Chunks/Passages in RAG (Retrieval Augmented Generation)

- 4️⃣ **Chunk Size (The Most Important Decision)**
- **Token-Based vs Character-Based**
- Most modern systems use **token-based chunking**, not characters.
- **Common Chunk Sizes**

Use Case	Chunk Size (tokens)
FAQs	150–300
Technical docs	300–500
Academic papers	500–800
Code	100–300
Legal / contracts	800–1,200

Chunks/Passages in RAG (Retrieval Augmented Generation)

- **5️⃣ Chunk Overlap (Context Preservation)**
- Information often spans boundaries. Overlap prevents losing meaning.
- **Example**
- Chunk 1: tokens 0–400
- Chunk 2: tokens 350–750
- Overlap: 50 tokens
- **Typical overlap values**

Chunk Size	Overlap
200	20–40
400	40–80
800	80–150

Chunks/Passages in RAG (Retrieval Augmented Generation)

Types of Chunking Strategies

1. Fixed-Size Chunking (Naive)

- Split every N tokens
- Fast and simple
- Often breaks semantic boundaries

✅ Good for demos

❌ Poor semantic quality

2. Recursive / Structural Chunking (Best Practice ✅)

Splits text **based on structure**, then enforces max length.

Hierarchy example:

Document
├— Section
│ ├— Subsection
│ │ ├— Paragraph

This preserves:

- Headings
- Paragraph meaning
- Logical flow
- ✅ Most common in production RAG

Chunks/Passages in RAG (Retrieval Augmented Generation)

3. Semantic Chunking

Uses embeddings or models to split by **semantic shifts** instead of size.

Example:

- Topic change detected → new chunk

✅ Excellent coherence
❌ Expensive and slower

Used in **advanced and research-grade RAG systems**.

4. Domain-Specific Chunking

Customized rules depending on data type:

Data Type	Chunk Strategy
Code	Function / class level
Research papers	Section + paragraph
Legal	Clause / article
Tables	Row groups
Logs	Time window

Chunks/Passages in RAG (Retrieval Augmented Generation)

7. Chunk Metadata (Often More Important Than Chunk Text)
Each chunk should carry **metadata**.

Common metadata fields


```
{
  "source": "ISO_Concrete_Standard.pdf",
  "section": "3.2 Definitions",
  "page": 12,
  "date": "2022",
  "author": "ISO",
  "topic": "concrete durability"
}
```


Metadata enables:

- Filtering (by date, section)
- Hybrid search
- Better reranking
- Citations

🧠 **Advanced RAG often retrieves by metadata first, embeddings second.**

Chunks/Passages in RAG (Retrieval Augmented Generation)

How Many Chunks Are Retrieved?

Typically **top-k chunks**:

k	Use
3–5	Simple QA
5–8	Technical answers
8–15	Complex synthesis

Beyond that:

- LLM degrades
- Hallucinations increase

✅ Better to retrieve fewer **high-quality** chunks.

Chunks/Passages in RAG (Retrieval Augmented Generation)

The **context window** is the **maximum number of tokens** an LLM can process **at once**, including:

- System instructions
- Conversation history
- Retrieved RAG chunks
- The user's question

If information is **outside the context window**, the model **cannot see or reason about it**.

What Consumes the Context Window?

A typical RAG prompt looks like this:

```
[System prompt]
[Safety / style instructions]
[Conversation history]
[Retrieved chunks]
[User question]
```

Token Budget Breakdown (Example: 8k model)

Component	Tokens
System prompt	300
Memory / chat history	1,500
RAG chunks (5 × 600)	3,000
User question	100
Response budget	2,000
Total	**~6,900**

In a RAG (Retrieval-Augmented Generation) prompt, the response budget is the portion of the model's context window reserved for the model's own generated answer. If you don't reserve enough tokens for the response:

- The answer gets cut off
- Reasoning chains are truncated
- Outputs become incomplete or incoherent

Chunks/Passages in RAG (Retrieval Augmented Generation)

9. Chunking vs Context Window

Model Context	Practical Retrieval
8k tokens	2–4 chunks
16k tokens	4–8 chunks
32k tokens	8–15 chunks

⚠️ Bigger context ≠ better answers if chunks are weak.

Best-Practice Checklist ✅

✓ Use **token-based sizing**
✓ Prefer **recursive / structural splitting**
✓ Add **overlap (10–20%)**
✓ Attach **rich metadata**
✓ Tune chunk size per domain
✓ Evaluate with **retrieval metrics**, not just LLM output

Vector Databases in RAG

1. What Is a Vector Database?

A **vector database** stores **embeddings** (numerical vectors) and allows **fast similarity search** over them.

In **RAG**, it is the component that answers the question:
"Which pieces of knowledge are most semantically related to this query?"
Instead of keyword matching, vector DBs use **semantic similarity**.

2. Why Vector Databases Are Central to RAG

LLMs **do not search** — they generate.
Vector databases **retrieve knowledge**.
RAG works because:

- Documents → embeddings
- Query → embedding
- Similar vectors ≈ similar meaning

Without a vector DB, RAG collapses into:

- Keyword search only (low recall)
- Or full-document stuffing (high cost + hallucinations)

Vector Databases in RAG

3. High-Level RAG Flow with a Vector DB

Offline (Indexing)

Documents

↓

Chunking

↓

Embedding Model

↓

Vectors + Metadata

↓

Vector Database

Online (Query)

User Query

↓

Embedding Model

↓

Similarity Search (Top-k)

↓

Retrieved Chunks

↓

LLM Answer

Similarity Search: How Retrieval Works

Metric	Used When
Cosine similarity ✅	Most common
Dot product	Normalized embeddings
Euclidean distance	Less common for text

Vector Databases Commonly Used in RAG

Open-Source

DB	Strength
FAISS	Fast, local, research
Chroma	Simple, RAG-focused
Milvus	Distributed, scalable
Weaviate	Hybrid search + schema

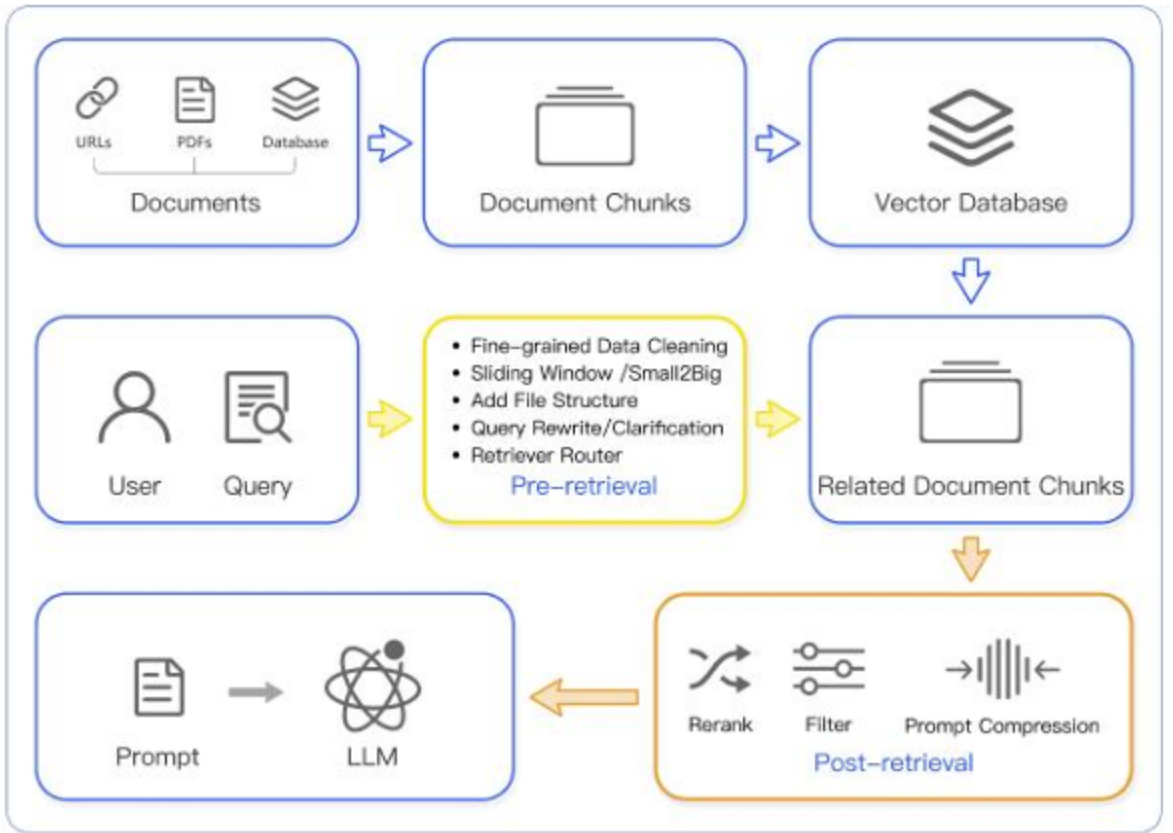
Managed / Cloud

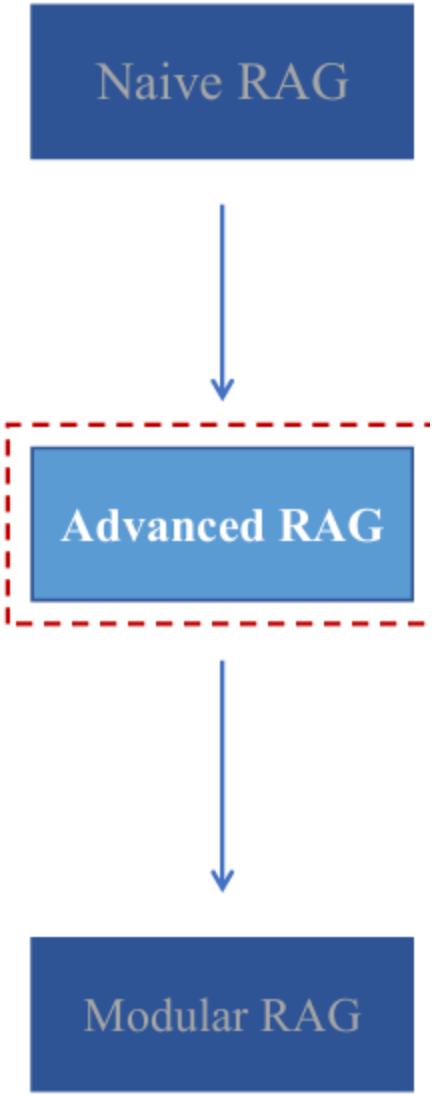
DB	Strength
Pinecone	Fully managed, robust
Qdrant	Strong filtering
Azure AI Search	Enterprise integration
OpenSearch	Keyword + vector

Advanced RAG

Index Optimization → Pre-Retrieval Process → Retrieval → Post-Retrieval Process → Genaration

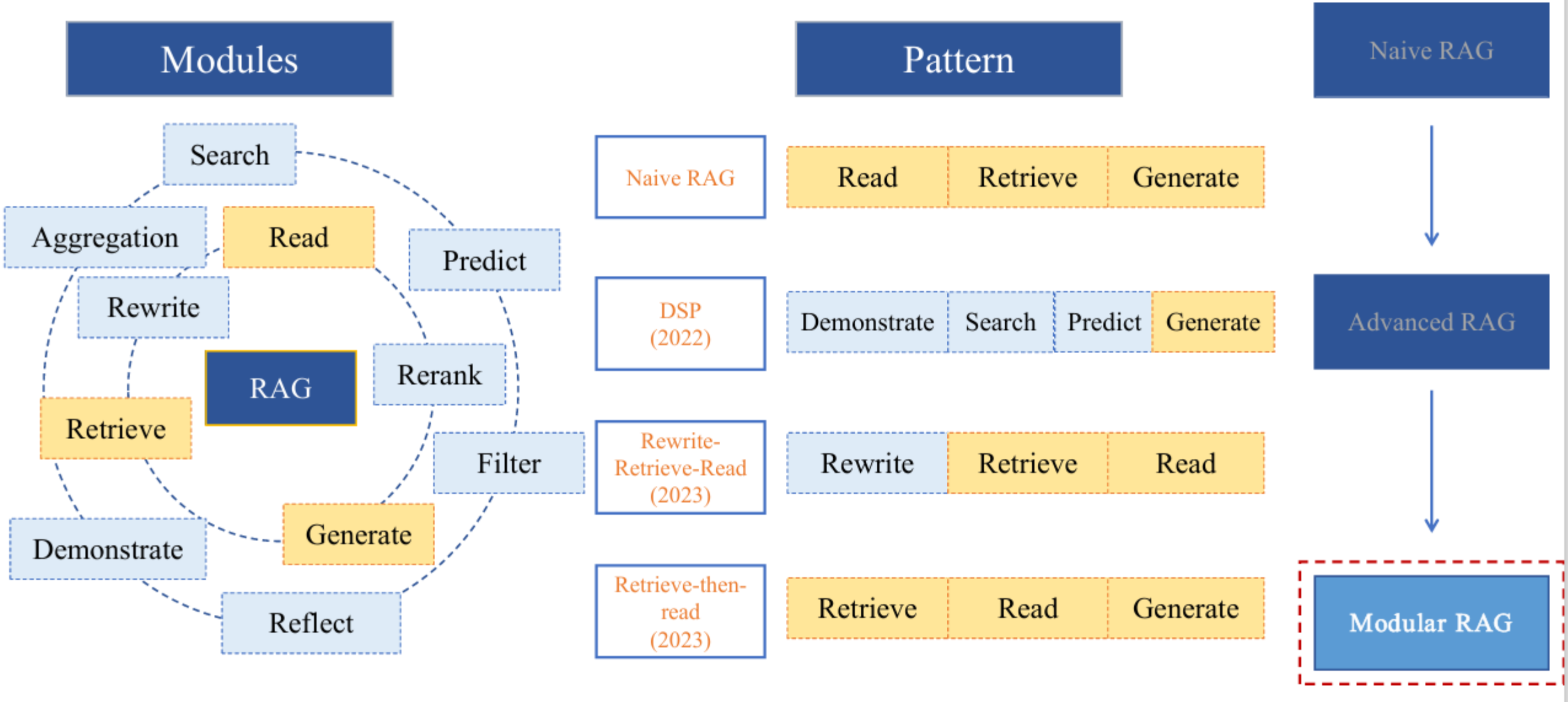
- **Optimizing Data Indexing:** sliding window, fine-grained segmentation、adding metadata
- **Pre-Retrieval Process:** retrieve routes, summaries, rewriting, and confidence judgment
- **Post-Retrieval Process:** reorder, filter content retrieval





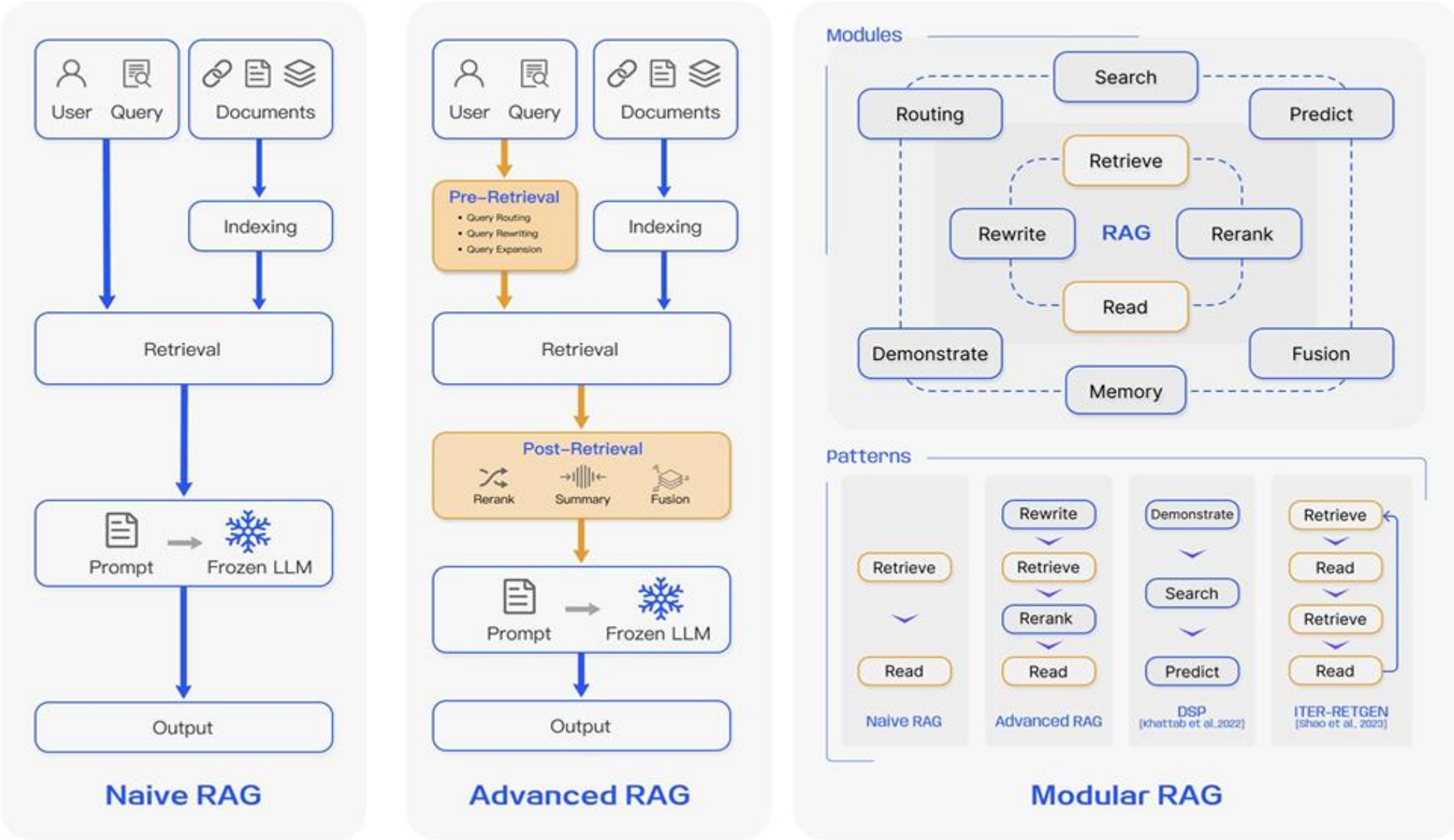
Source: Haofen Wang

Modular RAG



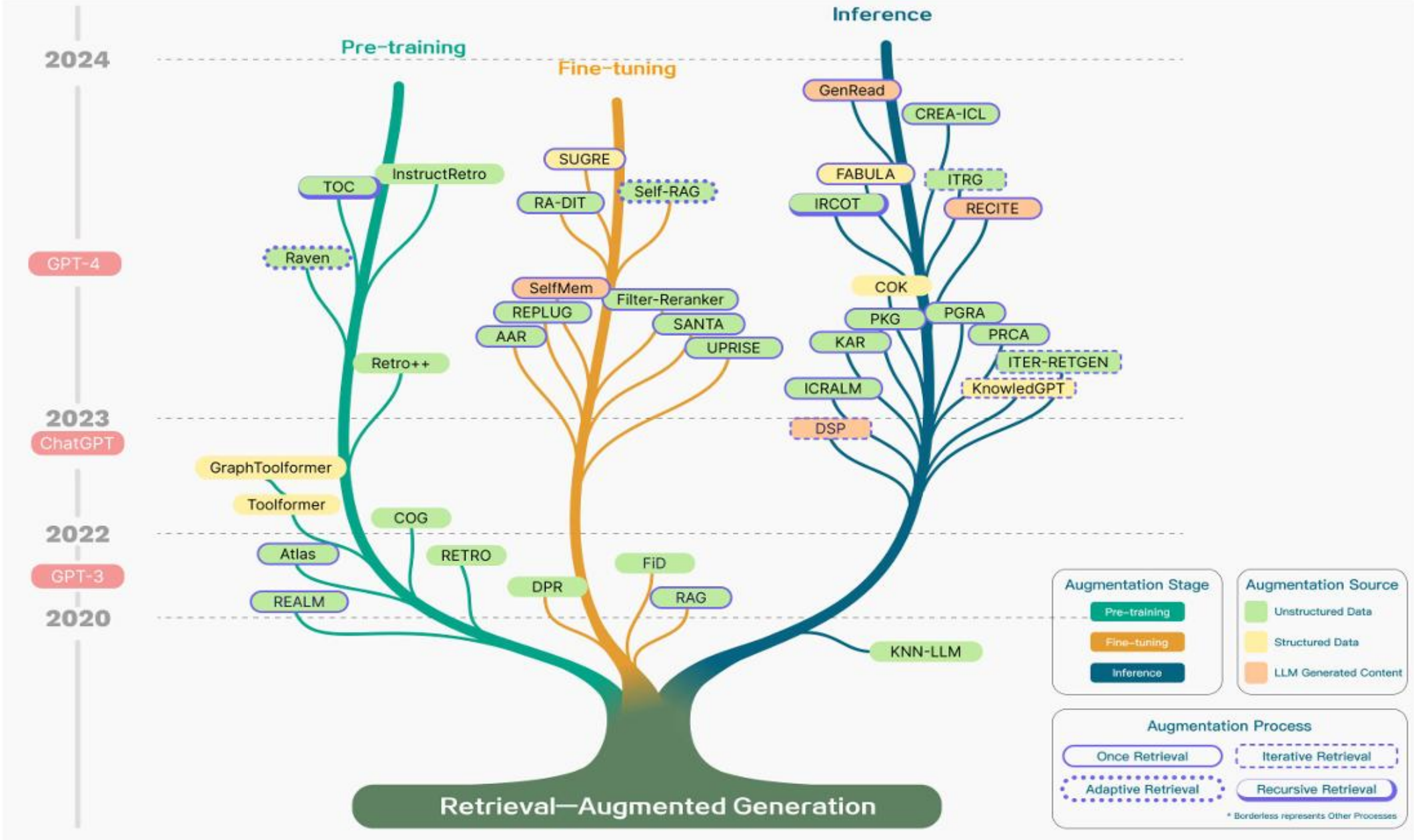
Source: Haofen Wang

Comparison of RAG Paradigms



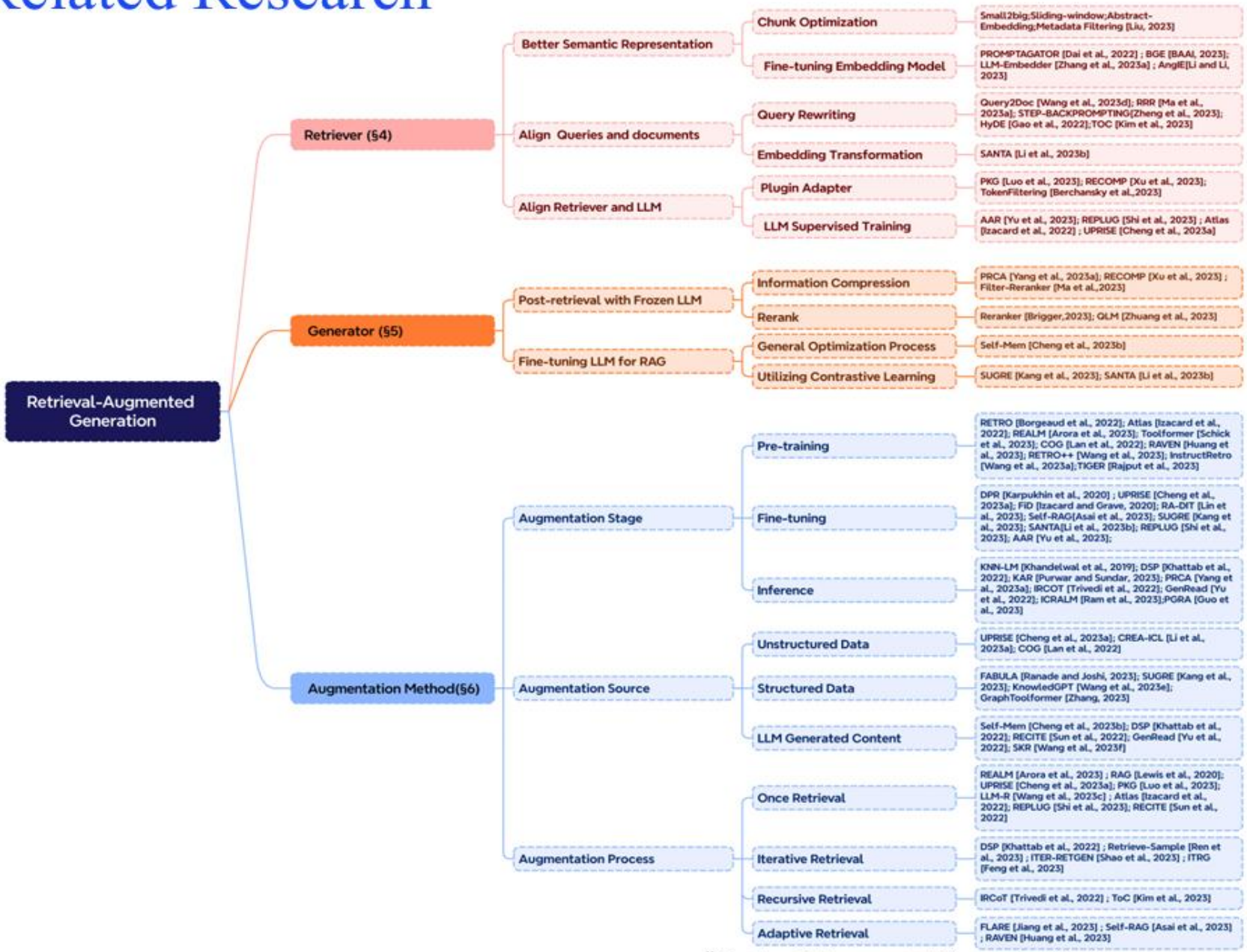
Source: Haofen Wang

Overview of RAG Development



Source: Haofen Wang

Summary of Related Research



《Retrieval-Augmented Generation for Large Language Models: A Survey》

Source: Haofen Wang

How to Evaluate the Effectiveness of RAG

EvaluationMethods

Independent Evaluation

Retriever
Evaluate the Quality of Text Blocks Retrieved by the Query
Metrics: MRP, Hit Rate, NDCG

Generation/Synthesis
Quality of Context Enhanced with Retrieved Documents Evaluation
Metrics: Context Relevance

End-to-End Evaluation

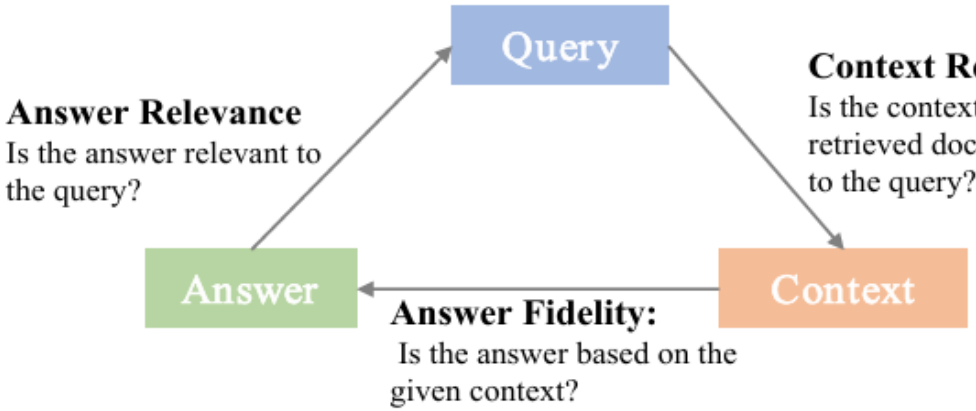
Evaluate the content ultimately generated by the model.

By generated conten
With labels： EM， Accuracy
Without labels: Fidelity, Relevance, Harmlessness

By evaluation method
Human evaluation
Automatic evaluation (LLM judge)

Key Metrics & Capabilities

Key Metrics



Key Capabilities

Noise Robustness
Can the model extract useful information from noisy documents?

Negative Rejection
When therequired knowledge is not exsiting in the retrieved documents, the answer should be refused.

Info Integration
Can the model answer complex questions that require integrating information from multiple documents?

Counterfactual Robustness
Can the model recognize the risk of known factual errors in the retrieved documents?

Assessment Framework

Use LLM as the adjudicator judge.

TruLens
RAGAS
ARES

Based on handwritten prompt

Synthetic dataset + Fine-tuning + Ranking using confidence intervals

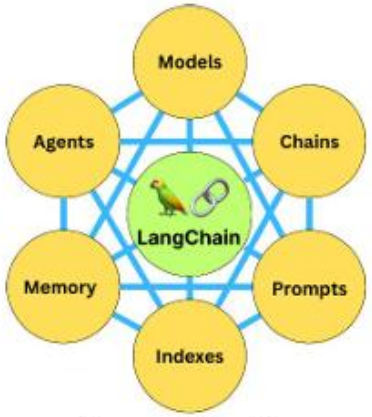
Evaluation

- Answer Fidelity
- Answer Relevance
- Contextual Relevance

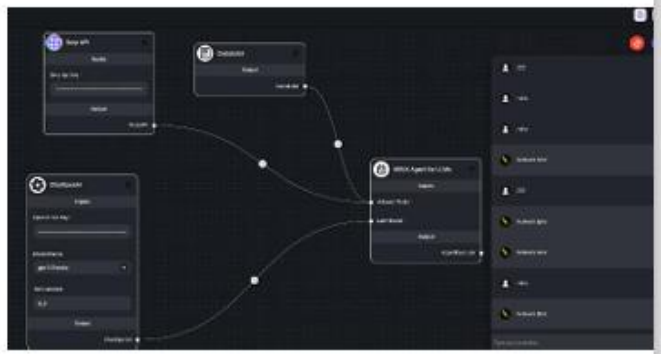
Source: Haofen Wang

Existing Tech Stack for RAG

Name	Pros	Cons
LangChain	Modular, full-featured	Inconsistent behavior ,API conceals details,complexity and low flexibility.
LlamaIndex	Focus on RAG	Requires combination use, low customization.
FlowiseAI	Easy to get started, visualized workflows.	Does not support complex scenarios.
AutoGen	Adapts to multi-agent scenarios.	Low efficiency, requires multiple rounds of dialogue.

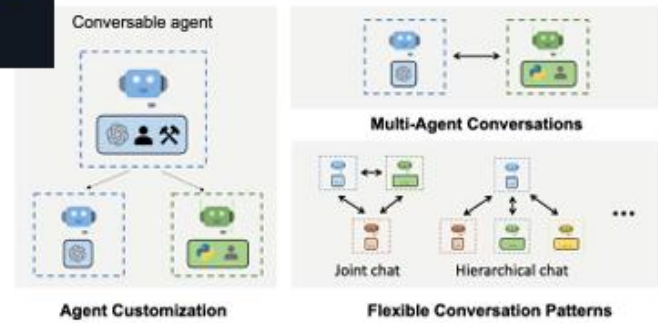


LangChain



FlowiseAI

LlamaIndex

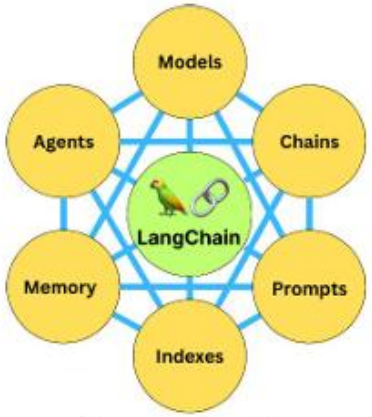


AutoGen

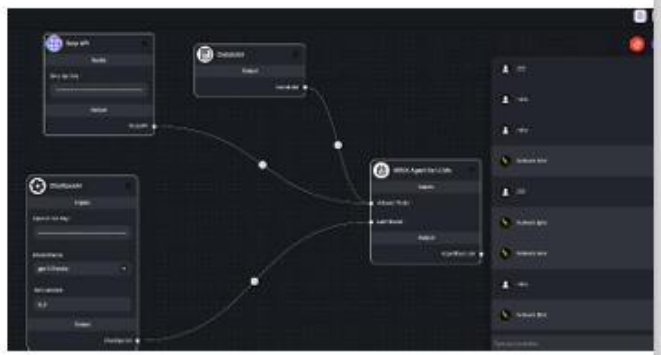
Source: Haofen Wang

Existing Tech Stack for RAG

Name	Pros	Cons
LangChain	Modular, full-featured	Inconsistent behavior ,API conceals details,complexity and low flexibility.
LlamaIndex	Focus on RAG	Requires combination use, low customization.
FlowiseAI	Easy to get started, visualized workflows.	Does not support complex scenarios.
AutoGen	Adapts to multi-agent scenarios.	Low efficiency, requires multiple rounds of dialogue.

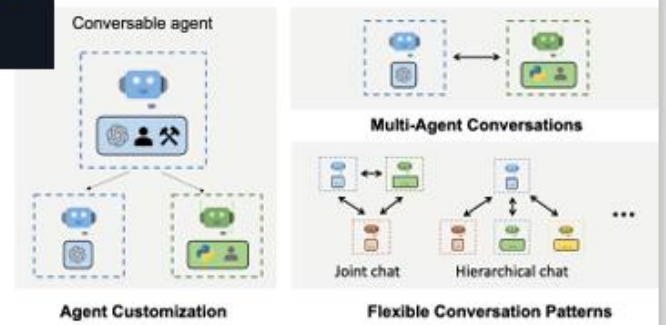


LangChain



FlowiseAI

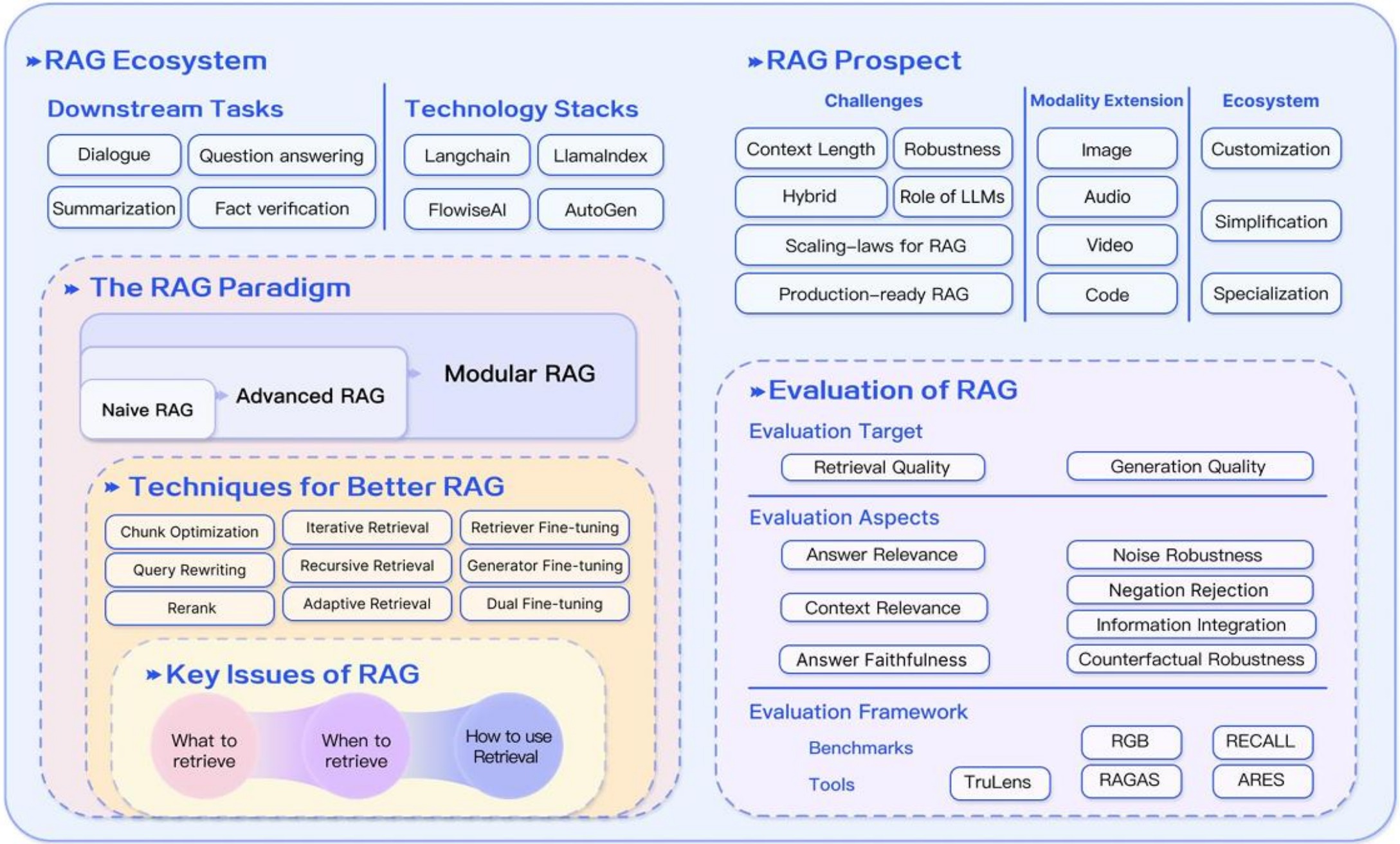
LlamaIndex



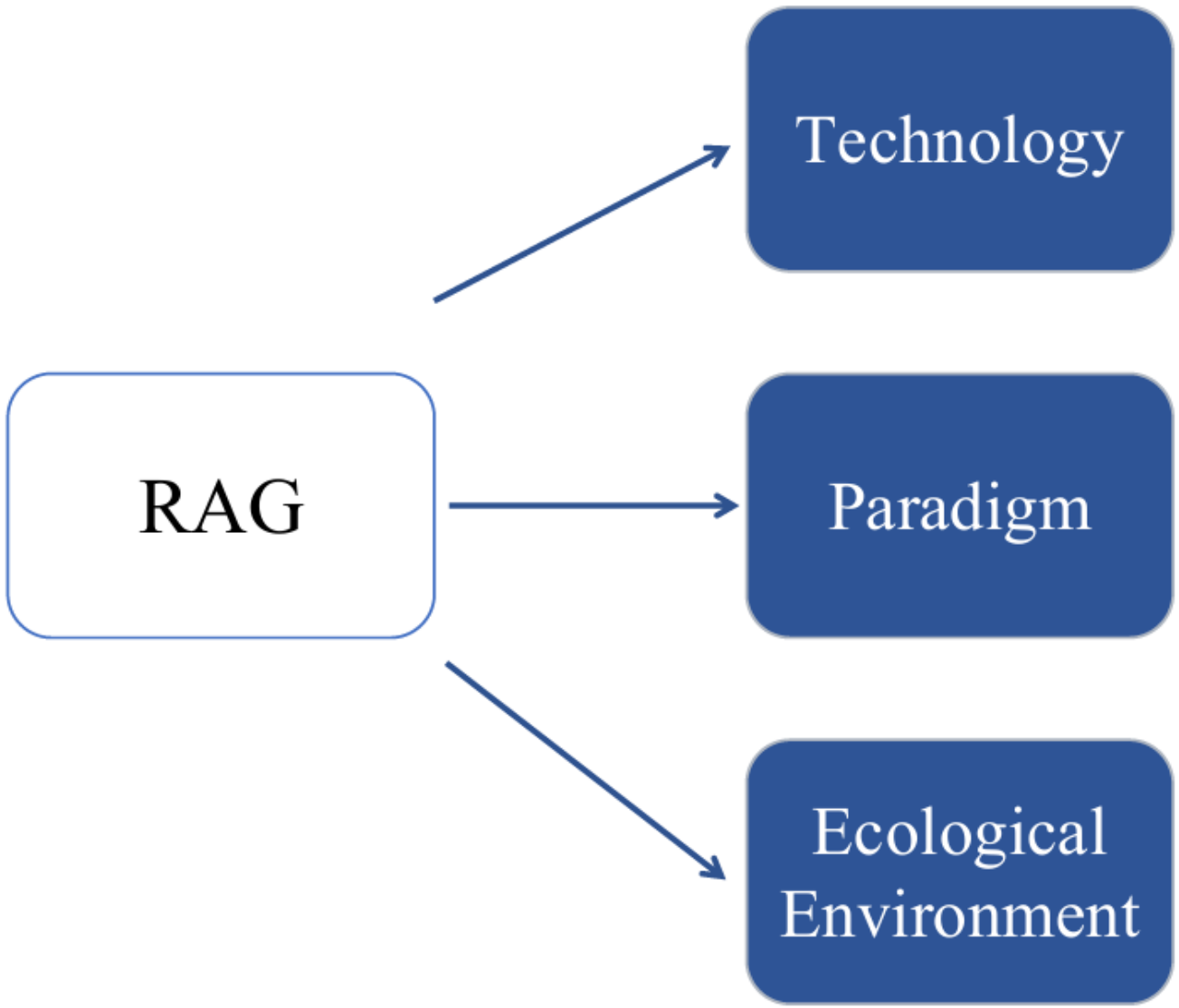
AutoGen

Source: Haofen Wang

Summary — The Framework of RAG



Summary —— Three Trends of RAG



- The Scaling Law of RAG Models
- How to Improve the Efficiency of Retrieving Large-scale Data
- Mitigation of Forgetting in Long-context Scenarios
- Enhancement of Multimodal Retrieval

- Modularity Will Become Mainstream
- Patterns for Module Organization Await Refinement
- Evaluation Systems Need to Evolve and Improve with Time

- Preliminary Formation of Toolchain Technology Stack
- One-stop Platform Still Requires Polishing
- Explosion of Enterprise-level Applications

Source: Haofen Wang

Prospects — Existing Challengs of RAG

Further address the challenges faced by RAG itself

Long context

- Retrieved content is excessive, exceeding window limit.
- The context is too long to result Lost in the Milddle.
- If the context window is not limited, is there still a need for RAG?

Coordination with FT

- How to simultaneously leverage the effects of RAG and FT.
- How do the two coordinate, how are they organized, is it in Pipeline, alternating, or end-to-end?

The role of LLMs

- LLM can be used for retrieval (LLM generation replaces retrieval, retrieving from LLM memory), for generation, and for evaluation. How to further explore the potential of LLM in RAG.

Robustness

- How to handle the incorrect content retrieved
- How to filter and verify the content retrieved.
- How to improve the model's resistance to toxicity and noise

Scaling Law

- Does the RAG model satisfy the Scaling Law
- Does RAG exhibit, or under what scenarios does it exhibit an Inverse Scaling Law

Engineering Practice

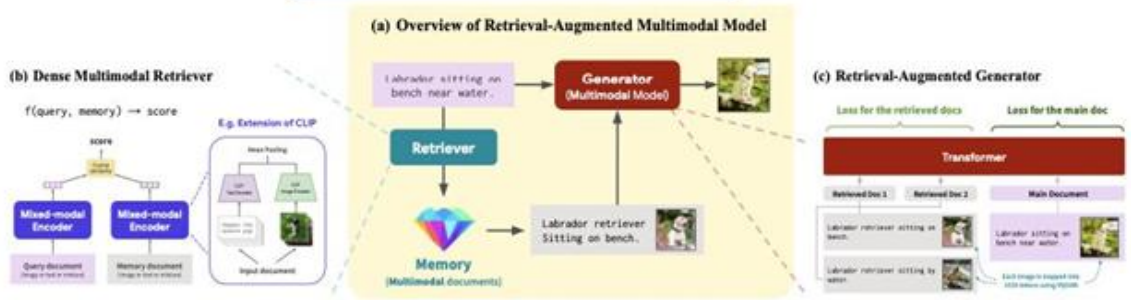
- How to reduce the latency of retrieving ultra-large-scale corpora.
- How to ensure that the content retrieved is not leaked by large models

Source: Haofen Wang

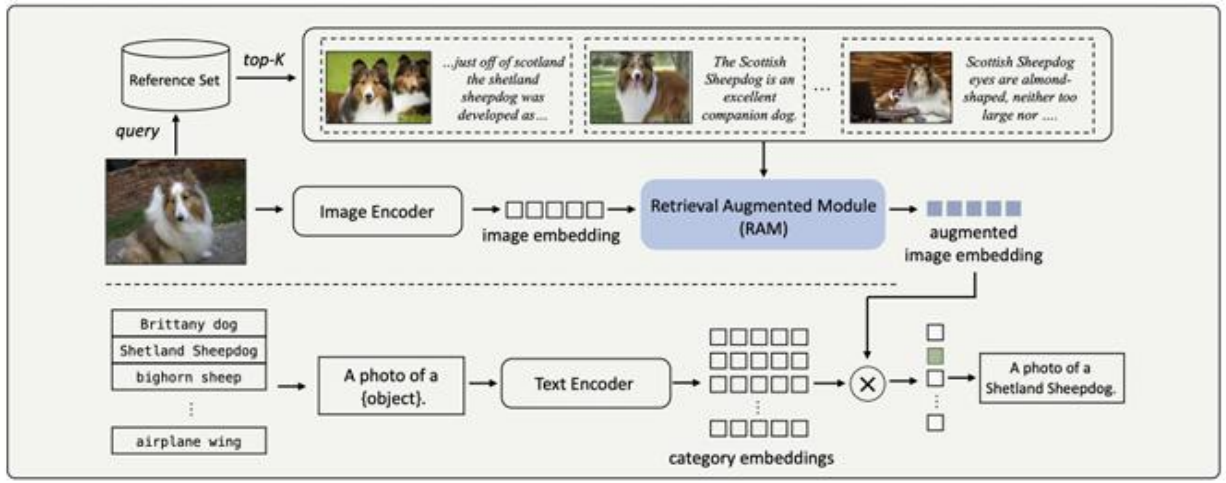
Prospects — Mult-Modality Extension

Transferring the concept of RAG from text to other modalities of data

Image

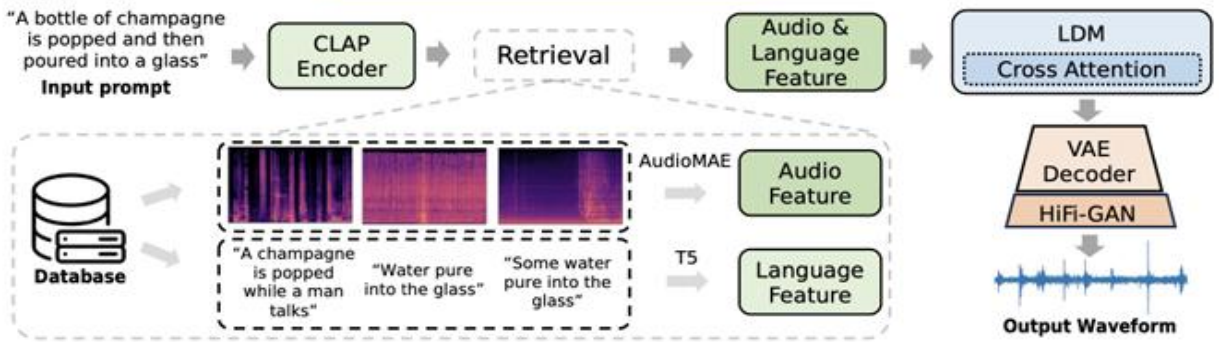


RA-CM3 [Yasunaga et al.,2023]



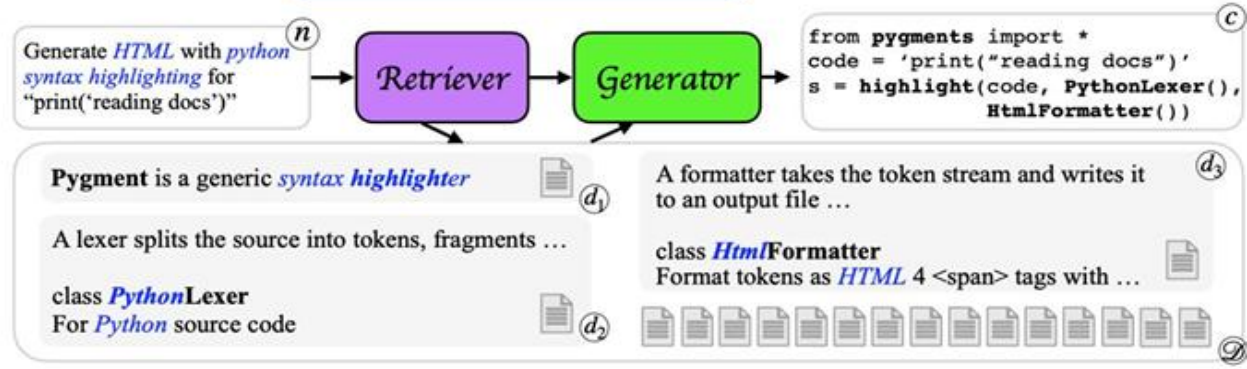
RA-CLIP [Xie et al.,2023]

Vedio



Re-AudioLDM [Yuan et al.,2023]

Code



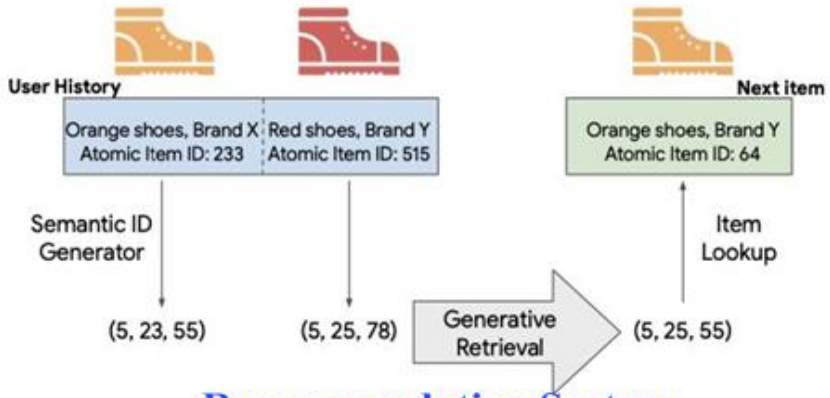
DocPrompting [Zhou et al.,2023]

Source: Haofen Wang

Prospects —— Development of RAG Ecosystem

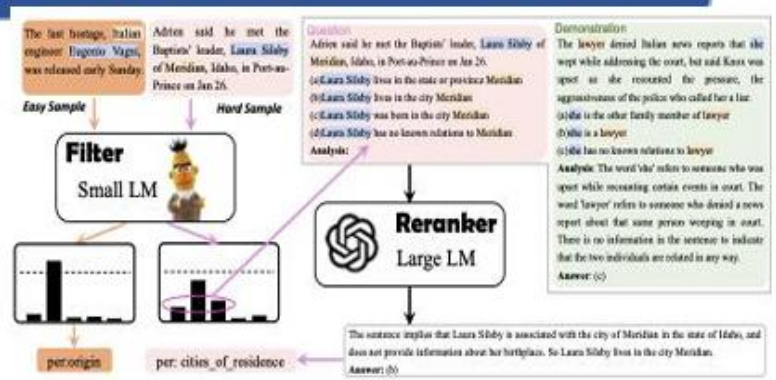
Further expand the downstream tasks of RAG and improve ecological construction

Downstream Task Development and Evaluation



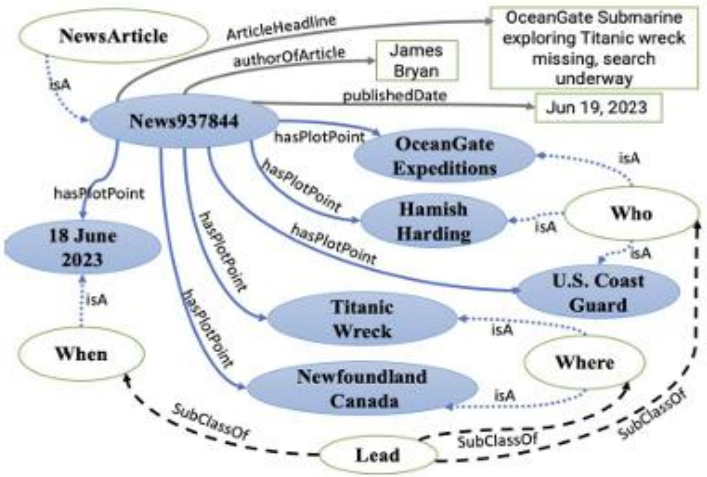
Recommendation System

| TIGER [Rajput et al.,2023]



Information extraction

| Filter- Rerank [Ma et al.,2023]



Report generation

| FABULA [Ranade et al.,2023]

Technology Stack Construction

- Customized function, meeting a variety of needs
- Simplified use, further reducing the barrier to entry.
- Specialized functions, gradually towards production environments.

Personal Knowledge Assistant Based on RAG

Open-source framework for production environments

Source: Haofen Wang

Reference

1. Alon, U. et al. Neuro-Symbolic Language Modeling with Automaton-augmented Retrieval.
2. Lewis, P. et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.
3. Guu, K., Lee, K., Tung, Z., Pasupat, P. & Chang, M.-W. REALM: Retrieval-Augmented Language Model Pre-Training. Preprint at http://arxiv.org/abs/2002.08909 (2020).
4. Dai, Z. et al. Promptagator: Few-shot Dense Retrieval From 8 Examples. Preprint at http://arxiv.org/abs/2209.11755 (2022).
5. Izacard, G. et al. Atlas: Few-shot Learning with Retrieval Augmented Language Models. Preprint at http://arxiv.org/abs/2208.03299 (2022).
6. Gao, L., Ma, X., Lin, J. & Callan, J. Precise Zero-Shot Dense Retrieval without Relevance Labels. Preprint at http://arxiv.org/abs/2212.10496 (2022).
7. Muennighoff, N., Tazi, N., Magne, L. & Reimers, N. MTEB: Massive Text Embedding Benchmark. in Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics 2014–2037 (Association for Computational Linguistics, 2023).
8. Ren, Y. et al. Retrieve-and-Sample: Document-level Event Argument Extraction via Hybrid Retrieval Augmentation. in Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) 293–306 (Association for Computational Linguistics, 2023).
9. Zhang, J. et al. ReAugKD: Retrieval-Augmented Knowledge Distillation For Pre-trained Language Models. in Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers) 1128–1136 (Association for Computational Linguistics, 2023). 10. Khattab, O. et al. Demonstrate-Search-Predict: Composing retrieval and language models for knowledge-intensive NLP. Preprint at http://arxiv.org/abs/2212.14024 (2023).
11. Cheng, X. et al. Lift Yourself Up: Retrieval-augmented Text Generation with Self Memory. Preprint at http://arxiv.org/abs/2305.02437 (2023).
12. Luo, Z. et al. Augmented Large Language Models with Parametric Knowledge Guiding. Preprint at http://arxiv.org/abs/2305.04757 (2023).
13. Shi, W. et al. REPLUG: Retrieval-Augmented Black-Box Language Models. Preprint at http://arxiv.org/abs/2301.12652 (2023).
14. Yu, Z., Xiong, C., Yu, S. & Liu, Z. Augmentation-Adapted Retriever Improves Generalization of Language Models as Generic Plug-In. Preprint at http://arxiv.org/abs/2305.17331 (2023).
15. Kang, M., Kwak, J. M., Baek, J. & Hwang, S. J. Knowledge Graph-Augmented Language Models for Knowledge-Grounded Dialogue Generation. Preprint at http://arxiv.org/abs/2305.18846 (2023).
16. Trivedi, H., Balasubramanian, N., Khot, T. & Sabharwal, A. Interleaving Retrieval with Chain-of-Thought Reasoning for Knowledge-Intensive Multi-Step Questions. Preprint at http://arxiv.org/abs/2212.10509 (2023).
17. Wang, L., Yang, N. & Wei, F. Learning to Retrieve In-Context Examples for Large Language Models. Preprint at http://arxiv.org/abs/2307.07164 (2023).
18. Li, Z. et al. Towards General Text Embeddings with Multi-stage Contrastive Learning. Preprint at http://arxiv.org/abs/2308.03281 (2023).
19. Ng, Y. et al. SimplyRetrieve: A Private and Lightweight Retrieval-Centric Generative AI Tool. Preprint at http://arxiv.org/abs/2308.03983 (2023).
20. Huang, J. et al. RAVEN: In-Context Learning with Retrieval Augmented Encoder-Decoder Language Models. Preprint at http://arxiv.org/abs/2308.07922 (2023).

Source: Haofen Wang

Reference

21. Zhu, Y. et al. Large Language Models for Information Retrieval: A Survey. Preprint at http://arxiv.org/abs/2308.07107 (2023).
22. Wang, X. et al. KnowledGPT: Enhancing Large Language Models with Retrieval and Storage Access on Knowledge Bases. Preprint at http://arxiv.org/abs/2308.11761 (2023).
23. Chen, J., Lin, H., Han, X. & Sun, L. Benchmarking Large Language Models in Retrieval-Augmented Generation. Preprint at http://arxiv.org/abs/2309.01431
24. Es, S., James, J., Espinosa-Anke, L. & Schockaert, S. RAGAS: Automated Evaluation of Retrieval Augmented Generation. Preprint at http://arxiv.org/abs/2309.15217 (2023).
25. Yoran, O., Wolfson, T., Ram, O. & Berant, J. Making Retrieval-Augmented Language Models Robust to Irrelevant Context. Preprint at http://arxiv.org/abs/2310.01558 (2023).
26. Feng, Z., Feng, X., Zhao, D., Yang, M. & Qin, B. Retrieval-Generation Synergy Augmented Large Language Models. Preprint at http://arxiv.org/abs/2310.05149 (2023).
27. Zheng, H. S. et al. Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models. Preprint at http://arxiv.org/abs/2310.06117 (2023).
28. Cheng, D. et al. UPRISE: Universal Prompt Retrieval for Improving Zero-Shot Evaluation. Preprint at http://arxiv.org/abs/2303.08518 (2023).
29. Wang, B. et al. InstructRetro: Instruction Tuning post Retrieval-Augmented Pretraining. Preprint at http://arxiv.org/abs/2310.07713 (2023).
30. Jiang, Z. et al. Active Retrieval Augmented Generation. Preprint at http://arxiv.org/abs/2305.06983 (2023).
31. Gou, Q. et al. Diversify Question Generation with Retrieval-Augmented Style Transfer. Preprint at http://arxiv.org/abs/2310.14503 (2023).
32. Ma, X., Gong, Y., He, P., Zhao, H. & Duan, N. Query Rewriting for Retrieval-Augmented Large Language Models. Preprint at http://arxiv.org/abs/2305.14283 (2023).
33. Yang, H. et al. PRCA: Fitting Black-Box Large Language Models for Retrieval Question Answering via Pluggable Reward-Driven Contextual Adapter. Preprint at http://arxiv.org/abs/2310.18347 (2023).
34. Kim, G., Kim, S., Jeon, B., Park, J. & Kang, J. Tree of Clarifications: Answering Ambiguous Questions with Retrieval-Augmented Large Language Models. Preprint at http://arxiv.org/abs/2310.14696 (2023).
35. Shao, Z. et al. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. Preprint at http://arxiv.org/abs/2305.15294 (2023).
36. Zhang, P., Xiao, S., Liu, Z., Dou, Z. & Nie, J.-Y. Retrieve Anything To Augment Large Language Models. Preprint at http://arxiv.org/abs/2310.07554 (2023).
37. Purwar, A. & Sundar, R. Keyword Augmented Retrieval: Novel framework for Information Retrieval integrated with speech interface. Preprint at http://arxiv.org/abs/2310.04205 (2023).
38. Lin, X. V. et al. RA-DIT: Retrieval-Augmented Dual Instruction Tuning. Preprint at http://arxiv.org/abs/2310.01352 (2023).
39. Yu, W. et al. Chain-of-Note: Enhancing Robustness in Retrieval-Augmented Language Models. Preprint at http://arxiv.org/abs/2311.09210 (2023).

Source: Haofen Wang

RAG Survey

- Most of the slides from
  - Our paper : https://arxiv.org/abs/2312.10997
  - Our GitHub: https://github.com/Tongji-KGLLM/RAG-Survey